BSc verslag TECHNISCHE WISKUNDE

Een adaptief Monte Carlo algoritme voor het schatten van de dominante eigenwaarde en eigenvector

(Engelse titel: An adaptive Monte Carlo algorithm for estimating the dominant eigenvalue and eigenvector)

Marnix Brands

Technische Universiteit Delft

Begeleider

Dr.ir. L.E. Meester

Overige commissieleden

Dr. B. van den Dries

Dr. J.J. Cai

Augustus 2019

Delft

Inhoudsopgave

Lijst met gebruikte symbolen

Symbool	Omschrijving	Sectie/Hoofdstuk
P	Algemene matrix	2
$\det(\cdot)$	Determinant	2
$\mathbb{E}(\cdot)$	Verwachting	3.1
$\bar{(\cdot)}$	Gemiddelde van argument	3.1
$s.e.(\cdot)$	Standard error	3.1
$\mathbf{P}$, $\mathbf{Q}$	Kansmaten $\mathbf{P}$, $\mathbf{Q}$	3.2
$\mathbb{E}^Q(\cdot)$	Verwachting onder kansmaat $\mathbf{Q}$	3.2
$Var^Q(\cdot)$	Variantie onder kansmaat $\mathbf{Q}$	3.2
L_n, L_τ	Likelihood ratio	3.2, 5
τ, T	Stoptijden behorende bij de Markovketen Y	4
Δ	Kerkhof	4
$\mathbf{1}_{\{\tau<T\}}$	Indicator van de gebeurtenis $\tau < T$	4
S	Toestandsruimte van de Markovketen Y	4
u^*	Exacte waarde eigenvector	4, 5
$\mathbb{R}^d_+$	$\{\boldsymbol{x} \in \mathbb{R}^d : x_i > 0 \quad \forall i\}$	5
n	Steekproefgrootte in Desai's algoritme	5
$W(i, \beta)$	Uitkomst simulatie $\beta^\tau L_\tau, Y_0 = i$ in Desai's algoritme	5
$Var(\cdot)$	Variantie van argument	6.1
$U(0, 1)$	standaard uniform verdeelde stochast	6.2
$u^{(m)}$	Schatting eigenvector na m iteraties	6.3
$\|\| \cdot \|\|$	Euclidische norm van het argument	6.3, 9
$< \cdot \| \cdot >$	Inwendig product	7.1
$O(\cdot)$	Grote-O-notatie	8.2

1 Inleiding

De rangschikking van webpagina's bepalen (de zogeheten GooglePageRank) en een vriendschapsverzoek ontvangen op Twitter hebben op het eerste gezicht niet veel met elkaar gemeen. Echter, de algoritmes die gebruikt worden om de rangschikking en het vriendschapsverzoek te realiseren, zijn wiskundig gezien identiek. Beide algoritmes zijn gebaseerd op het vinden van de dominante eigenwaarde van een matrix. Het vinden van eigenwaarden en de bijbehorende eigenvectoren van een matrix is een bekend probleem binnen de wiskunde en hoe makkelijk het probleem te formuleren is, hoe moeilijk het vaak is om het probleem exact op te lossen. Vaak is het zo dat de eigenwaarden en eigenvectoren alleen benaderd kunnen worden aan de hand van een numeriek algoritme. Er zijn meerdere algoritmes die de eigenwaarden van een matrix (of een aantal daarvan) kunnen benaderen. Vaak hangt de keuze van het algoritme af van het type matrix (bijvoorbeeld symmetrisch of niet-symmetrisch) en of alle eigenwaarden of slechts een enkele eigenwaarde benaderd moet worden. De algoritmes die gebruikt worden om eigenwaarden te benaderen zijn in het algemeen deterministisch van aard. Dit houdt in dat voor een gegeven matrix altijd dezelfde benadering van de eigenwaarden wordt gevonden zodra de begintoestand van het algoritme gelijk blijft. Dit betekent dus dat zogeheten randomness geen rol speelt bij dit soort algoritmes.
In dit onderzoek zal gekeken worden naar een algoritme voor het benaderen van de dominante eigenwaarde en bijbehorende eigenvector waarbij randomness wel en rol speelt. Dit algoritme is beschreven door P.Y. Desai in zijn proefschrift (Desai, 2001) en vanaf nu zal dit algoritme dan ook worden aangeduid als Desai's algoritme. De vraag die dan opspeelt, is of Desai's algoritme een goed alternatief is voor de bestaande (deterministische) numerieke methoden. De hoofdvraag in dit onderzoek luidt dan ook als volgt:

"Is Desai's algoritme voor het vinden van de dominante eigenwaarde en eigenvector een concurrent voor de power methode?"

In dit onderzoek zal Desai's algoritme worden beschreven dat gebruikt kan worden om de dominante eigenwaarde en bijbehorende eigenvector te benaderen. Met behulp van Monte Carlo simulatie is het mogelijk om de randomness te simuleren die in Desai's algoritme aanwezig is.
Eerst zal een korte introductie worden gegeven over wat Monte Carlo simulatie precies inhoudt. Daarnaast zal ook een belangrijke techniek binnen de Monte Carlo simulatie, genaamd importance sampling, worden toegelicht. Dit wordt gedaan, aangezien ook in Desai's algoritme importance sampling wordt toegepast. Vervolgens zal worden verteld hoe Desai's algoritme precies tot stand is gekomen en hierbij zal de basis worden gelegd voor de werking van het algoritme. Zodra de werking van het algoritme is beschreven, zal ook een complexiteitsanalyse worden gemaakt om de rekentijd en nauwkeurigheid van het algoritme te onderzoeken. Om te bepalen of Desai's algoritme een goed alternatief is voor de bestaande numerieke methoden, zal het algoritme naast de bekende power methode worden gelegd. Om deze twee algoritmes met elkaar te kunnen ver-

gelijken, zullen er een aantal testmatrices in de praktijk worden gesimuleerd. Hierbij zullen de convergentiesnelheid en de totale rekentijd van de algoritmes naast elkaar gezet worden, om een antwoord te kunnen geven op de hoofdvraag van dit onderzoek. Naast het vergelijken met de power methode, zal ook voor een aantal factoren specifiek onderzocht worden of deze van invloed zijn op convergentiesnelheid in Desai's algoritme.

2 Probleemstelling

In dit onderzoek staat het vinden van de dominante eigenwaarde en bijbehorende eigenvector van een matrix centraal. Het vinden van de eigenwaarden van een matrix is een bekend probleem binnen de wiskunde en wordt hier nog even kort samengevat. Laat P een $d \times d$ matrix zijn en laat λ de waarde zijn met bijbehorende vector u die aan de volgende vergelijking voldoen:

$$Pu = \lambda u.$$

Dan wordt λ een eigenwaarde en u de bijbehorende eigenvector van de matrix P genoemd. Bovenstaande vergelijking kan ook genoteerd worden als volgt:

$$(P - \lambda I)u = 0.$$

De eigenwaarden van de matrix P kunnen gevonden worden door de vergelijking

$$det(P - \lambda I) = 0$$

op te lossen. Hierbij wordt met $det(P)$ de determinant van de matrix P bedoeld. Zodra deze determinant wordt uitgewerkt, resulteert dit in een polynoom van graad d voor λ en dit polynoom wordt ook wel de karakteristieke vergelijking van de matrix P genoemd. De hoofdstelling van de algebra geeft ons dat er in totaal d oplossingen zijn (waarbij dubbele nulpunten als verschillend worden beschouwd) van deze karakteristieke vergelijking. Dit betekent dat een $d \times d$ matrix dus maximaal d verschillende eigenwaarden kan hebben. De stelling van Abel-Ruffini (voor een algemene beschrijving van deze stelling zie het artikel van Edixhoven (2013)) zegt dat voor een polynoom van graad $d \geq 5$ geen algemene oplossingsmethode bestaat. In deze gevallen zijn de exacte oplossingen vaak niet te achterhalen en kunnen de eigenwaarden alleen benaderd worden.
De matrices die in dit onderzoek aan bod komen, zijn altijd niet-negatief en irreducibel (voor de definitie van een irreducibele matrix zie sectie A.1). Zodra een matrix aan deze voorwaarden voldoet, volgt uit de bekende Perron-Frobenius stelling (zie, bijvoorbeeld, sectie 1.4 van Minc (1988)) dat er een unieke strikt positieve eigenwaarde λ en een bijbehorende strikt positieve eigenvector u^* bestaat zodanig dat geldt :

$$Pu^* = \lambda u^*.$$

Uit de stelling volgt ook dat de Perron-Frobenius eigenwaarde λ de dominante eigenwaarde van de matrix P is ($|\lambda| > |\mu|$ voor alle andere eigenwaarden μ van P). Het doel is om met behulp van Desai's algoritme een goede benadering te vinden voor λ en u^*.

3 Introductie Monte Carlo simulatie

3.1 Wat is Monte Carlo simulatie?

Monte Carlo simulatie is een simulatietechniek die vaak wordt gebruikt in toepassingen waarbij onzekerheden een rol spelen. Hierbij wordt een proces meerdere keren gesimuleerd, waarin de onzekerheidsfactor wordt meegenomen (dit gebeurd vaak door middel van het random genereren van getallen tijdens de simulatie). Dit zorgt ervoor dat er rekening wordt gehouden met de onzekerheden die aanwezig zijn. Het resultaat van deze verzameling simulaties is een scala aan mogelijke uitkomsten van het proces. Aan de hand van deze verzameling uitkomsten kan dan ook de verwachte uitkomst van het proces worden benaderd. Monte Carlo simulatie wordt bijvoorbeeld gebruikt om optieprijzen voor aandelen of de levensduur van onderdelen in machines te benaderen. Ook voor deterministische problemen kan eventueel Monte Carlo simulatie worden gebruikt om oplossingen te benaderen. Zo kan π met behulp van Monte Carlo simulatie zeer nauwkeurig benaderd worden (Madras, 2002). De precieze werking van Monte Carlo simulatie zal hier nu kort besproken worden.

Stel je wilt een waarde, genoteerd met a, bepalen. De eerste stap in een Monte Carlo simulatie is om deze waarde te schrijven als de verwachting van een stochastische variabele X die gesimuleerd kan worden:

$$\mathbb{E}(X) = a. \tag{1}$$

De volgende stap in een Monte Carlo simulatie is om een aantal simulaties uit te voeren met de stochast X. Hierbij wordt het aantal simulaties ook wel het aantal replicaties genoemd en vaak genoteerd met de letter M. We simuleren dus M onafhankelijke trekkingen $X_1, X_2, ..., X_M$. Vervolgens nemen we het gemiddelde van deze trekkingen en noteren dat als volgt:

$$\bar{X}_M = \frac{1}{M} \sum_{i=1}^{M} X_i.$$

De waarde voor $\bar{X}_M$ vormt dan de schatting voor de waarde van a en uit de wet van de grote aantallen volgt dat voor $M \to \infty$ geldt dat $\bar{X}_M \to a$. Dit betekent dat in het algemeen de schatting van de waarde a beter wordt naarmate het aantal replicaties toeneemt. Nu is een schatting zonder een betrouwbaarheidsinterval (of een andere maat voor de foutmarge) niks waard, aangezien alleen een schatting geen informatie bevat over de nauwkeurigheid. Om een uitspraak te kunnen doen over de nauwkeurigheid van de schatting wordt de standard error berekend. Definieer $\sigma^2 = Var(X)$, de variantie van de stochast X. De standard error is dan de standaarddeviatie van het steekproefgemiddelde $\bar{X}_M$ en ziet er als volgt uit:

$$s.e.(\bar{X}_M) = \sqrt{Var(\bar{X}_M)} = \sqrt{\frac{Var(X)}{M}} = \frac{\sigma}{\sqrt{M}}.$$

Indien de variantie van X niet bekend is, wordt vaak de steekproefvariantie berekend aan de hand van de trekkingen $X_1, X_2, ..., X_M$ en genoteerd met $\hat{\sigma}^2$. De standard error wordt dan berekend als volgt:

$$s.e.(\bar{X}_M) = \frac{\hat{\sigma}}{\sqrt{M}}.$$

De standard error kan ook worden worden gebruikt om een betrouwbaarheidsinterval voor a te construeren. Zo wordt een 95% betrouwbaarheidsinterval voor a gegeven door (dit volgt uit de centrale limietstelling):

$$\left[\bar{X}_M - 1.96 \cdot s.e.(\bar{X}_M) \quad , \quad \bar{X}_M + 1.96 \cdot s.e.(\bar{X}_M)\right]$$

3.2 Importance sampling: een introductie

Bij Monte Carlo simulatie bestaan er vaak meerdere stochastische variabelen waarmee gesimuleerd kan worden. Deze stochastische variabelen hebben dan allen gemeen dat de verwachting gelijk is aan de gezochte waarde a. De varianties van de stochasten kunnen echter wel verschillen. Uit de definitie van de standard error is te zien dat deze kleiner wordt als de variantie van de stochast ook afneemt. Een belangrijk punt dat dan ook bij Monte Carlo simulatie komt kijken, is het kiezen van een te simuleren stochast X waar de variante klein van is.
Ook bestaan er technieken om een stochast X, waarmee gesimuleerd kan worden, om te vormen tot een andere geschikte stochast met een kleinere variantie. Dit soort technieken worden dan ook variantie-reducerende technieken genoemd. Een belangrijk voorbeeld hiervan is importance sampling. Deze techniek kan leiden tot een goede variantie-reductie (mits goed toegepast) en de werking hiervan zal hieronder toegelicht worden.
Laat X de stochastische variabele zijn waarmee gesimuleerd kan worden en $f(x)$ de bijbehorende dichtheidsfunctie. De kansmaat die behoort bij $f(x)$ noteren we met $\mathbf{P}$. De verwachting van X onder $\mathbf{P}$ zullen we weergeven met $\mathbb{E}^{\mathbf{P}}(X)$ (waarvoor dus geldt dat $\mathbb{E}^{\mathbf{P}}(X) = a$) en de variantie wordt genoteerd als $Var^{\mathbf{P}}(X)$. Het idee achter importance sampling is om over te stappen op een andere kansmaat, genoteerd met $\mathbf{Q}$. De dichtheidsfunctie van X onder kansmaat $\mathbf{Q}$ zal worden aangegeven met $g(x)$. De verwachting van X kan dan als volgt geschreven worden:

$$\begin{aligned} \mathbb{E}^{\mathbf{P}}(X) &= \int_{-\infty}^{\infty} x f(x) dx \\ &= \int_{-\infty}^{\infty} x \frac{f(x)}{g(x)} \cdot g(x) dx \\ &= \mathbb{E}^{\mathbf{Q}}(X \cdot L) \end{aligned} \tag{2}$$

waarbij

$$L(x) = \frac{f(x)}{g(x)} \tag{3}$$

de likelihood ratio wordt genoemd. In plaats van X te simuleren onder de kansmaat $\mathbf{P}$, wordt bij importance sampling $X \cdot L$ gesimuleerd onder de nieuwe kansmaat $\mathbf{Q}$. Dit leidt dan tot variantie-reductie zodra $Var^{\mathbf{Q}}(X \cdot L) < Var^{\mathbf{P}}(X)$. Uit de berekening bij (2) is te zien dat er een probleem ontstaat zodra $g(x) = 0$. Een voorwaarde voor de keuze van $g(x)$ is dan ook dat $g(x)$ positief moet zijn wanner $f(x)$ positief is (nog preciezer, wanneer $xf(x)$ niet negatief is). Dit betekent dus dat gebeurtenissen met positieve kans onder maat $\mathbf{P}$ ook een positieve kans hebben onder maat $\mathbf{Q}$.

De variantie van $X \cdot L(X)$ is klein wanneer

$$x \cdot L(x) = x\frac{f(x)}{g(x)} \approx constant$$

en de variantie is zelfs nul als

$$x \cdot L(x) = x\frac{f(x)}{g(x)} = constant.$$

De kansmaat $\mathbf{Q}$ (ofwel de dichtheidsfunctie $g(x)$) waarvoor de variantie gelijk is aan nul wordt de zogeheten zero-variance verdeling genoemd. In theorie zou met het gebruik van deze verdeling het volstaan om slechts één simulatie uit te voeren. Echter deze verdeling is in de praktijk niet te gebruiken, aangezien deze verdeling gebruik maakt van de te bepalen waarde (in dit geval zal dus de waarde a nodig zijn voor het bepalen van de zero-variance verdeling). Toch wordt de zero-variance verdeling vaak berekend, omdat dit veel informatie bevat over wat een goede kansmaat $\mathbf{Q}$ zou kunnen zijn waar wel mee gesimuleerd kan worden.

3.3 Importance sampling voor Markovketens

In sectie 5 wordt Desai's algoritme beschreven dat gebruik maakt van Monte Carlo simulatie om de dominante eigenwaarde en eigenvector van een matrix te vinden. Hierbij wordt importance sampling toegepast op een discrete Markovketen en in deze sectie zal worden beschreven hoe importance sampling in dat geval werkt. Laat $X = f(Y_0, Y_1, ..., Y_n)$ een stochastische variabele zijn waarvan de uitkomst wordt bepaald door de Markovketen $Y = (Y_n : n \geq 0)$ met bijbehorende overgangsmatrix P (we nemen aan dat de begintoestand bekend is en dat er dus geen beginverdeling wordt gegeven). Laat $P(i, j)$ de overgangskans zijn van toestand i naar toestand j. Voor de Markovketen Y hebben we dan het

volgende:

$$\begin{aligned}\mathbb{P}(Y_1 = y_1, Y_2 = y_2, ..., Y_n = y_n | Y_0 = y_0) &= P(y_0, y_1) \cdot P(y_1, y_2)...P(y_{n-1}, y_n) \\ &= \prod_{i=0}^{n-1} P(y_i, y_{i+1})\end{aligned}$$

De verwachting van X onder de overgangsmatrix P en begintoestand $Y_0 = y_0$ wordt genoteerd als $\mathbb{E}^P_{y_0}(X)$. De verwachting krijgt dan de volgende uitdrukking:

$$\begin{aligned}\mathbb{E}^P_{y_0}(X) &= \mathbb{E}^P_{y_0}(f(Y_0, Y_1, ..., Y_n)) \\ &= \sum_{y_1,...,y_n} f(y_0, y_1, ..., y_n) \cdot \mathbb{P}(Y_1 = y_1, Y_2 = y_2, ..., Y_n = y_n | Y_0 = y_0) \\ &= \sum_{y_1,...,y_n} f(y_0, y_1, ..., y_n) \cdot \prod_{i=0}^{n-1} P(y_i, y_{i+1})\end{aligned}$$

Ook in deze situatie kan importance sampling worden toegepast. Het idee is om over te gaan op een andere overgangsmatrix, genoteerd met Q. Op eenzelfde manier als bij (2) vinden we dan het volgende resultaat:

$$\begin{aligned}\mathbb{E}^P_{y_0}(X) &= \sum_{y_1,...,y_n} f(y_0, y_1, ..., y_n) \cdot \prod_{i=0}^{n-1} P(y_i, y_{i+1}) \\ &= \sum_{y_1,...,y_n} f(y_0, y_1, ..., y_n) \cdot \frac{\prod_{i=0}^{n-1} P(y_i, y_{i+1})}{\prod_{i=0}^{n-1} Q(y_i, y_{i+1})} \cdot \prod_{i=0}^{n-1} Q(y_i, y_{i+1}) \\ &= \mathbb{E}^Q_{y_0}(X \cdot L_n)\end{aligned} \tag{4}$$

waarbij L_n ook weer de likelihood ratio wordt genoemd. We zien dus dat in het geval van het simuleren onder een Markovketen, de likelihood ratio die ontstaat bij importance sampling wordt gegeven door:

$$L_n = \frac{\prod_{i=0}^{n-1} P(Y_i, Y_{i+1})}{\prod_{i=0}^{n-1} Q(Y_i, Y_{i+1})} = \prod_{i=0}^{n-1} \frac{P(Y_i, Y_{i+1})}{Q(Y_i, Y_{i+1})}. \tag{5}$$

Net als in de vorige sectie bij de keuze voor de dichtheidsfunctie $g(x)$, geldt ook nu dat de overgangsmatrix Q niet willekeurig gekozen kan worden. Uit de berekening bij (4) is te zien dat er een probleem ontstaat als $Q(y_i, y_{i+1}) = 0$ voor een zekere waarde van i. De voorwaarde waaraan de matrix Q dus moet voldoen is dat $Q(y_i, y_{i+1})$ positief is wanneer $P(y_i, y_{i+1})$ positief is. Dit betekent dat een overgang met positieve kans onder matrix P ook een positieve kans heeft onder matrix Q.

4 Totstandkoming van Desai's algoritme

Om tot een werkend Monte Carlo algoritme te komen, is de eerste stap om de gevraagde waarde te schrijven als uitkomst van de verwachting van een te simuleren stochastische variabele (zie vergelijking (1)). Dit houdt dus in dat voor Desai's algoritme zowel voor de dominante eigenwaarde als de bijbehorende eigenvector zo'n dergelijke representatie gevonden moet worden. Het vinden van deze representaties is gebaseerd op het artikel van Glynn (1996).
Laat $A = (A(i,j) : i,j \in S)$ een niet-negatieve, irreducibele matrix zijn waarvan de dominante eigenwaarde λ en bijbehorende eigenvector u^* bepaald moeten worden. We bepalen eerst de grootste rijsom van de matrix A:

$$t = \max_{i \in S} \sum_{j=1}^{d} A(i,j).$$

Vervolgens gaan we over op de matrix $P = (P(i,j) : i,j \in S)$ waarvoor geldt

$$P(i,j) = \frac{A(i,j)}{t}.$$

Dit zorgt ervoor dat P een irreducibele substochastische matrix is. (voor de definitie van een substochastische matrix zie sectie A.2). Zonder verlies van algemeenheid mogen we in het vervolg aannemen dat de matrix P op deze manier genormaliseerd is.
De volgende stap is om de matrix P om te zetten in een stochastische matrix. Dit wordt gedaan door de toestandsruimte S uit te breiden met een extra toestand Δ. Deze toestand Δ wordt ook wel het "kerkhof"genoemd. Laat $\tilde{S} = S \cup \Delta$ de uitgebreide toestandsruimte zijn. Definieer nu de stochastische matrix $\tilde{P} = (\tilde{P}(i,j) : i,j \in \tilde{S})$ als volgt:

$$\tilde{P}(i,j) = \begin{cases} P(i,j), & i,j \in S \\ 1 - \sum_{j \in S} P(i,j), & i \in S, j = \Delta \\ 1, & i = j = \Delta \end{cases}$$

Aangezien $\tilde{P}$ een stochastische matrix is, kan deze gezien worden als een overgangsmatrix van een discrete Markovketen. Bij deze overgangsmatrix $\tilde{P}$ definiëren we dan ook de Markovketen $Y = (Y_n : n \geq 0)$ met toestandsruimte $\tilde{S}$. Laat

$$T = \inf\{n \geq 0 : Y_n = \Delta\}$$

de eerste keer zijn dat de Markovketen in toestand Δ terecht komt. Verder kies een vaste terugkeertoestand $k \in S$ en laat

$$\tau = \inf\{n \geq 1 : Y_n = k\}$$

de eerste keer zijn (begintoestand Y_0 niet meegerekend) dat de Markovketen in deze toestand k terecht komt. Definieer $\mathbb{E}_i(\cdot) = \mathbb{E}(\cdot | Y_0 = i)$. Voor de overzichtelijkheid in het vervolg zullen we de volgende functie definiëren:

$$v(i) = \mathbb{E}_i(\beta^{-\tau} \mathbf{1}_{\{\tau < T\}}). \tag{6}$$

De belangrijke aanname die vervolgens wordt gebruikt, luidt als volgt:

Aanname. *Er bestaat een $\beta_* > 0$ zodanig dat*

$$\mathbb{E}_k(\beta_*^{-\tau} \cdot \mathbf{1}_{\{\tau<T\}}) = 1$$

Hierbij is $\mathbf{1}_{\{\tau<T\}}$ een indicator stochastische variabele die de waarde 1 aanneemt als $\tau < T$ en 0 anders. De motivatie om deze aanname te maken, wordt in sectie A.3 beschreven. Echter voor het begrijpen van de verdere werking van het algoritme is deze informatie niet noodzakelijk. De aanname geeft dus dat

$$v(k) = \mathbb{E}_k(\beta_*^{-\tau}\mathbf{1}_{\{\tau<T\}}) = 1. \tag{7}$$

De volgende stap is om $v(i)$ uit te schrijven in termen van de matrix P. Hiervoor zal eerst geconditioneerd worden op $Y_1 = j$ voor $j \in S$ (voor $j = \Delta$ geldt $v(\Delta) = 0$). We zullen het geval $Y_1 = k$ eerst apart bekijken. Voor $Y_1 = k$ geldt dat $\tau = 1$ en aangezien de waarde voor τ dan bekend is (en dus ook de uitkomst van $\mathbf{1}_{\{\tau<T\}}$), kan de Markovketen gestopt worden. Voor $Y_1 = k$ geldt dan het volgende voor $v(i)$:

$$\begin{aligned} v(i) &= \mathbb{E}_i\Big(\mathbb{E}_i\Big(\beta^{-\tau} \cdot \mathbf{1}_{\{\tau<T\}}|Y_1 = k\Big)\Big) \\ &= \mathbb{E}_i\Big(\beta^{-1}P(i,k)\Big) \\ &= \beta^{-1}P(i,k) \end{aligned} \tag{8}$$

Voor de andere toestanden $j \in S$ kan de Markov-eigenschap worden toegepast om een uitdrukking te vinden voor $v(i)$. Deze eigenschap zegt dat de zodra geconditioneerd wordt op Y_1, dat Y_1 alle informatie bevat over de volgende overgang van de Markovketen (en Y_0 er dus niet meer toe doet). Deze eigenschap zal nu gebruikt worden om een uitdrukking te vinden voor $v(i)$ wanneer geconditioneerd wordt op $Y_1 = j$ voor $j \in S$:

$$\begin{aligned} v(i) &= \mathbb{E}_i\Big(\mathbb{E}_i\Big(\beta^{-\tau} \cdot \mathbf{1}_{\{\tau<T\}}|Y_1 = j\Big)\Big) \\ &= \mathbb{E}_i\Big(\beta^{-1}P(i,j)\mathbb{E}_j(\beta^{-\tau} \cdot \mathbf{1}_{\{\tau<T\}})\Big) \quad (*) \\ &= \beta^{-1}\sum_{j\in S, j\neq k} P(i,j)\mathbb{E}_j(\beta^{-\tau} \cdot \mathbf{1}_{\{\tau<T\}}) + \beta^{-1}P(i,k) \\ &= \beta^{-1}\sum_{j\in S, j\neq k} P(i,j)v(j) + \beta^{-1}P(i,k) \end{aligned} \tag{9}$$

waarbij $(*)$ de stap aangeeft waar de Markov-eigenschap is toegepast. Nu maken we gebruik van de aanname dat $v(k) = 1$ voor $\beta = \beta_*$ en dit resulteert in het

volgende

$$\begin{aligned} v(i) &= \beta_*^{-1} \sum_{j \neq k} P(i,j)v(j) + \beta_*^{-1} P(i,k)v(k) \qquad (\text{want} \quad v(k) = 1) \\ &= \beta_*^{-1} \sum_{j} P(i,j)v(j) \end{aligned}$$

De gevonden uitdrukking omschrijven geeft dan het volgende resultaat:

$$\begin{aligned} \beta_* v(i) &= \sum_j P(i,j)v(j) \\ \beta_* v &= Pv \end{aligned} \tag{10}$$

Hieruit volgt, als β_* de oplossing is van vergelijking (7), dat $\beta_* = \lambda$ en $v = u^*$. We hebben nu dus dat de vergelijkingen (6) en (7) een stochastische representatie vormen van de dominante eigenwaarde λ en eigenvector u^* behorende bij de matrix P. Aan de hand van deze stochastische representatie zal in de volgende sectie Desai's algoritme worden beschreven om de eigenwaarde λ en eigenvector u^* te benaderen.

5 Werking Desai's algoritme

Aan de hand van de vergelijkingen (6) en (7) zal nu Desai's algoritme worden beschreven om de eigenwaarde λ en eigenvector u^* te benaderen. Het algoritme maakt gebruik van importance sampling (zie sectie 3.2 en 3.3 voor een korte uitleg hiervan). Hierbij wordt de overgangsmatrix $\tilde{P}$ vervangen door een nieuwe overgangsmatrix Q. Laat u een (strikt positieve) schatting van de eigenvector u^* zijn. Dan wordt de nieuwe overgangsmatrix $Q_u = (Q_u(i,j) : i,j \in \tilde{S})$ gegeven door:

$$Q_u(i,j) = \begin{cases} \frac{P(i,j)u(j)}{\sum_{k\in S} P(i,k)u(k)}, & i,j \in S \\ 0, & i \in S, j = \Delta \\ 0, & i \in \Delta, j \in S \\ 1, & i = j = \Delta \end{cases} \tag{11}$$

Uit de vergelijkingen (6) en (7) is te zien dat de waarden van τ en T nodig zijn om $\beta^{-\tau}\mathbf{1}_{\{\tau<T\}}$ te kunnen bepalen. Echter uit de eigenschappen van de matrix Q_u volgt dat $\mathbb{P}(Y_n = \Delta | Y_{n-1} = i) = 0$ voor alle $i \in S$ en alle $n \in \mathbb{N}$. Dit houdt in dat het kerkhof nooit bereikt kan worden zodra de begintoestand Y_0 zich in S bevindt. Hieruit volgt ook dat $\mathbb{P}(\tau < T) = 1$ en dat de indicatorfunctie $\mathbf{1}_{\{\tau<T\}}$ altijd gelijk is aan 1. In het vervolg zullen we deze indicatorfunctie dan ook weglaten. Aangezien de waarde voor τ bepaald moet worden, zal de Markovketen Y worden gesimuleerd onder de overgangsmatrix Q_u totdat de terugkeertoestand k is bereikt. Zodra deze toestand bereikt is, wordt de simulatie gestopt en de volgende output gegenereerd:

$$W(i,\beta) = \beta^{-\tau} \prod_{n=0}^{\tau-1} \frac{P(Y_n, Y_{n+1})}{Q_u(Y_n, Y_{n+1})} \qquad Y_0 = i \tag{12}$$

waarbij

$$L_\tau = \prod_{n=0}^{\tau-1} \frac{P(Y_n, Y_{n+1})}{Q_u(Y_n, Y_{n+1})} \tag{13}$$

de likelihood ratio is die ontstaat zodra importance sampling wordt toegepast (zie vergelijking (5) uit sectie 3.3).
De eerste stap is om een schatting te maken van de eigenwaarde β_* en deze schatting zal genoteerd worden met $\hat{\beta}$. Hierbij wordt de schatter $W(k,\hat{\beta})$ n keer onafhankelijk gesimuleerd, waarbij k de gekozen terugkeertoestand is. Vervolgens kunnen we $v(k) = \mathbb{E}_k(\beta^{-\tau}\mathbf{1}_{\{\tau<T\}})$ schatten aan de hand van het steekproefgemiddelde:

$$\tilde{W}(k,\hat{\beta}) = \frac{1}{n}\sum_{l=1}^{n} \hat{\beta}^{-\tau_l} L_{\tau_l}$$

waarbij τ_l de waarde is voor τ in simulatie l van de Markovketen. Het aantal simulaties n wordt ook wel de steekproefgrootte genoemd. De gevonden functie wordt vervolgens gelijk gesteld aan 1:

$$\tilde{W}(k,\hat{\beta}) = \frac{1}{n}\sum_{l=1}^{n} \hat{\beta}^{-\tau_l} L_{\tau_l} = 1 \tag{14}$$

en het oplossen van deze functie geeft dan de schatting $\hat{\beta}$ van de eigenwaarde β_*.
Zodra $\hat{\beta}$ berekend is, kan voor iedere andere begintoestand $i \in S$ de functie $W(i, \hat{\beta})$ worden gesimuleerd. Ook hier wordt de schatter $W(i, \hat{\beta})$ met een steekproefgrootte van n gesimuleerd en kunnen we $v(i) = \mathbb{E}_i(\beta^{-\tau}\mathbf{1}_{\{\tau<T\}})$ schatten aan de hand van het steekproefgemiddelde:

$$\tilde{W}(i, \hat{\beta}) = \frac{1}{n}\sum_{l=1}^{n} W(i, \hat{\beta}).$$

Deze steekproefgemiddelden vormen vervolgens de schatting voor de eigenvector u^* (dus $\tilde{W}(i, \hat{\beta})$ vormt een schatting voor de component $u^*(i)$). Deze schatting wordt vervolgens de beginschatting van de volgende iteratie van het algoritme.

In het algoritme dat hierboven is beschreven, wordt de huidige schatting van u^* gebruikt als beginschatting van de volgende iteratie. Dit houdt in dat de overgangsmatrix Q_u in de volgende iteratie wordt aangepast aan de beste schatting van de huidige iteratie. Het algoritme leert dus als het ware van de vorige iteraties en past daar de de overgangsmatrix Q_u op aan. Een algoritme van deze vorm wordt daardoor ook wel een adaptief algoritme genoemd.
De volgende beste schatting in het algoritme ontstaat door met de huidige beste schatting het algoritme te doorlopen. Dit duidt er op dat de volgende beste schatting alleen afhangt van de huidige beste schatting. Met andere woorden, de reeks van beste schattingen is een Markovketen. Laat X_n de beste schatting zijn van de eigenvector voor iteratie n. Dan is reeks $X = (X_n : n \geq 0)$ een Markovketen. De toestandsruimte van deze Markovketen bevat de uitkomsten van deze beste schattingen en dus is deze toestandsruimte $\mathbb{R}_+^d$ (herinner dat uit de Perron-Frobenius stelling volgt dat de eigenvector altijd strikt positief is), waarbij d de dimensie van de matrix P is. De Markovketen heeft dus een continue uitkomstruimte.
De belangrijke eigenschappen waarvan we hopen dat de Markovketen X_n deze bezit, is dat X_n convergeert naar u^* voor $n \to \infty$ en dat u^* een vast punt van de keten is. Dit laatste betekent dat zodra $X_n = u^*$ voor een zekere $n \in \mathbb{N}$, dat ook $X_{n+1} = u^*$, $X_{n+2} = u^*$ enzovoort. In sectie 6.3 wordt de convergentie van het algoritme besproken en in sectie A.4 wordt bewezen dat u^* inderdaad een vast punt is van de Markovketen X_n.
In Desai's algoritme wordt importance sampling gebruikt om over te gaan op een nieuwe overgangsmatrix Q_u. De keuze voor deze matrix Q_u ziet er op het eerste gezicht misschien willekeurig uit, maar is gebaseerd op de zogeheten zero-variance kansmaat (zie sectie 3.2). In de volgende sectie zal de keuze voor deze overgangsmatrix Q_u toegelicht worden.

6 Enkele kenmerken Desai's algoritme uitgelicht

6.1 Waarom deze keuze voor de matrix Q_u?

In Desai's algortime wordt importance sampling toegepast om over te gaan op een nieuwe overgangsmatrix Q_u. Men probeert deze matrix zo te kiezen dat de variantie van de te simuleren stochast

$$W(i,\beta) = \beta^{-\tau} \prod_{n=0}^{\tau-1} \frac{P(Y_n, Y_{n+1})}{Q_u(Y_n, Y_{n+1})} = \beta^{-\tau} L_\tau \qquad\qquad Y_0 = i$$

klein is en in het optimale geval gelijk aan nul. De keuze voor de matrix Q_u in Desai's algoritme zal nu worden toegelicht. Desai heeft deze toelichting in zijn proefschrift (Desai, 2001) beknopt verteld en zal hier met meer tussenstappen beschreven worden.
Het idee is om de variantie (geconditioneerd op de begintoestand Y_0) van de stochastische variabele $W(i,\beta)$ te formuleren. Om deze variantie te verkrijgen, zal eerst de verwachting van $W(i,\beta)$ moeten worden bepaald. Zodra de variantie van $W(i,\beta)$ berekend is, zal daarna gekeken worden voor welke matrix Q_u deze variantie minimaal is.
Vergelijking (6) geeft dat

$$v(i) = \mathbb{E}_i^P(\beta^{-\tau}\mathbf{1}_{\{\tau<T\}}) = \mathbb{E}_i^Q(\beta^{-\tau}L_\tau) = \mathbb{E}^Q(W(i,\beta)).$$

Alle berekeningen in deze sectie zullen betrekking hebben tot kansmaat Q_u. We zullen daarom $\mathbb{E}$ in plaats van $\mathbb{E}^Q$ gebruiken voor de overzichtelijkheid.
We beginnen dus met het bepalen van de verwachting van $W(i,\beta)$. Hiervoor conditioneren we eerst op Y_1:

$$\begin{aligned}
\mathbb{E}(W(i,\beta)|Y_1 = j) &= \mathbb{E}(\beta^{-\tau}L_\tau|Y_0 = i, Y_1 = j) \\
&= \mathbb{E}\Big(\beta^{-\tau}\prod_{n=0}^{\tau-1}\frac{P(Y_n,Y_{n+1})}{Q(Y_n,Y_{n+1})}|Y_0 = i, Y_1 = j\Big) \\
&= \beta^{-1}\frac{P(Y_0,Y_1)}{Q(Y_0,Y_1)} \cdot \mathbb{E}\Big(\beta^{-\tau}\prod_{n=1}^{\tau-1}\frac{P(Y_n,Y_{n+1})}{Q(Y_n,Y_{n+1})}|Y_1 = j\Big) \\
&= \beta^{-1}\frac{P(i,j)}{Q(i,j)} \cdot \mathbb{E}\Big(\beta^{-\tau}\prod_{n=0}^{\tau-1}\frac{P(Y_n,Y_{n+1})}{Q(Y_n,Y_{n+1})}|Y_0 = j\Big) \quad (*) \\
&= \beta^{-1}\frac{P(i,j)}{Q(i,j)} \cdot \mathbb{E}(W(j,\beta)) \\
&= \beta^{-1}\frac{P(i,j)}{Q(i,j)} \cdot v(j)
\end{aligned}$$

waarbij $(*)$ aangeeft dat de Markov-eigenschap daar is toegepast. Vervolgens de wet van totale verwachting toepassen om de volgende uitdrukking te krijgen

voor de verwachting van $W(i,\beta)$:

$$\begin{aligned}\mathbb{E}(W(i,\beta)) &= \mathbb{E}\Big(\mathbb{E}(W(i,\beta)|Y_1=j)\Big)\\ &= \mathbb{E}\Big(\beta^{-1}\frac{P(i,j)}{Q(i,j)}\cdot v(j)\Big)\\ &= \sum_{j=1}^{d}\beta^{-1}\frac{P(i,j)}{Q(i,j)}\cdot v(j)\cdot Q(i,j) \qquad (15)\\ &= \beta^{-1}\sum_{j=1}^{d}P(i,j)v(j)\end{aligned}$$

De volgende stap is om de variantie van $W(i,\beta)$ te achterhalen. De variantie van $W(i,\beta)$ zullen we als volgt noteren:

$$\begin{aligned}z(i) &:= Var^{Q}\Big(W(i,\beta)\Big)\\ &= \mathbb{E}\Big(W(i,\beta)^2\Big) - \mathbb{E}\Big(W(i,\beta)\Big)^2\\ &= \mathbb{E}\Big(W(i,\beta)^2\Big) - v(i)^2\end{aligned}$$

Conditioneren op Y_1 geeft de volgende conditionele variantie van $W(i,\beta)$:

$$\begin{aligned}Var\Big(W(i,\beta)|Y_1=j\Big) &= \mathbb{E}\Big(W(i,\beta)^2|Y_1=j\Big) - \mathbb{E}\Big(W(i,\beta)|Y_1=j\Big)^2\\ &= \mathbb{E}\Big(\beta^{-2\tau}\prod_{n=0}^{\tau-1}\frac{P^2(Y_n,Y_{n+1})}{Q^2(Y_n,Y_{n+1})}|Y_0=i,Y_1=j\Big) - \Big(\beta^{-1}\frac{P(i,j)}{Q(i,j)}\cdot v(j)\Big)^2\\ &= \beta^{-2}\frac{P^2(Y_0,Y_1)}{Q^2(Y_0,Y_1)}\mathbb{E}\Big(\beta^{-2\tau}\prod_{n=1}^{\tau-1}\frac{P^2(Y_n,Y_{n+1})}{Q^2(Y_n,Y_{n+1})}|Y_1=j\Big) - \Big(\beta^{-1}\frac{P(i,j)}{Q(i,j)}\cdot v(j)\Big)^2\\ &= \beta^{-2}\frac{P^2(i,j)}{Q^2(i,j)}\mathbb{E}\Big(\beta^{-2\tau}\prod_{n=0}^{\tau-1}\frac{P^2(Y_n,Y_{n+1})}{Q^2(Y_n,Y_{n+1})}|Y_0=j\Big) - \Big(\beta^{-1}\frac{P(i,j)}{Q(i,j)}\cdot v(j)\Big)^2\\ &= \beta^{-2}\frac{P^2(i,j)}{Q^2(i,j)}\Big(\mathbb{E}(W(j,\beta))^2 - v(j)^2\Big)\\ &= \beta^{-2}\frac{P^2(i,j)}{Q^2(i,j)}z(j)\end{aligned}$$

Om de variantie (geconditioneerd op Y_0) van $W(i,\beta)$ te berekenen, maken we gebruik van de wet van de totale variantie. Deze wet geeft dat de variantie kan worden geschreven als

$$Var\Big(W(i,\beta)\Big) = Var\Big(\mathbb{E}(W(i,\beta)|Y_1=j)\Big) + \mathbb{E}\Big(Var(W(i,\beta)|Y_1=j)\Big)$$

en een bewijs hiervoor is te vinden in sectie A.5. Voor $z(i) = Var(W(i,\beta))$ geldt dan:

$$z(i) = Var\Big(\beta^{-1}\frac{P(i,j)}{Q(i,j)} \cdot v(j)\Big) + \mathbb{E}\Big(\beta^{-2}\frac{P^2(i,j)}{Q^2(i,j)}z(j)\Big)$$

We bekijken de twee termen apart. De linker term kan worden geschreven als volgt:

$$\begin{aligned} Var\Big(\beta^{-1}\frac{P(i,j)}{Q(i,j)}v(j)\Big) &= \mathbb{E}\left(\beta^{-2}\frac{P^2(i,j)}{Q^2(i,j)}v(j)^2\right) - \mathbb{E}\left(\beta^{-1}\frac{P(i,j)}{Q(i,j)}v(j)\right)^2 \\ &= \beta^{-2}\sum_{j=1}^{d}\frac{P^2(i,j)}{Q(i,j)}v(j)^2 - \mathbb{E}\Big(\mathbb{E}(W(i,\beta)|Y_1 = j)\Big)^2 \\ &= \beta^{-2}\sum_{j=1}^{d}\frac{P^2(i,j)}{Q(i,j)}v(j)^2 - \mathbb{E}\Big((W(i,\beta)\Big)^2 \\ &= \beta^{-2}\sum_{j=1}^{d}\frac{P^2(i,j)}{Q(i,j)}v(j)^2 - v(i)^2 \end{aligned}$$

Voor de rechter term vinden we op dezelfde manier als bij (15) de volgende uitdrukking:

$$\mathbb{E}\Big(\beta^{-2}\frac{P^2(i,j)}{Q^2(i,j)}z(j)\Big) = \beta^{-2}\sum_{j=1}^{d}\frac{P^2(i,j)}{Q(i,j)}z(j)$$

en dus kan $z(i)$ als volgt geschreven worden:

$$z(i) = \beta^{-2}\sum_{j=1}^{d}\frac{P^2(i,j)}{Q(i,j)}v(j)^2 - v(i)^2 + \beta^{-2}\sum_{j=1}^{d}\frac{P^2(i,j)}{Q(i,j)}z(j).$$

Definieer vervolgens de volgende twee functies:

$$f(i) = \beta^{-2}\sum_{j=1}^{d}\frac{P^2(i,j)}{Q(i,j)}v^2(j) - v^2(i) \tag{16}$$

$$A(i,j) = \begin{cases} \beta^{-2}\frac{P^2(i,j)}{Q(i,j)}, & Q(i,j) > 0 \\ 0, & Q(i,j) = 0 \end{cases}$$

Dan geldt voor de conditionele variantie $z(i)$ dat

$$z(i) = f(i) + (Az)_i. \tag{17}$$

Noteren we $\mathbf{z}$ als de vector van de conditionele varianties van de schatters $W(i,\beta)$ voor iedere begintoestand $i \in S$, dan voldoet $\mathbf{z}$ aan de volgende vergelijking:

$$\mathbf{z} = f + A\mathbf{z}.$$

Het volgende doel is nu om een uitdrukking voor de overgangsmatrix Q_u te vinden waarbij $\mathbf{z}$ minimaal is. Voor het vinden van een zero-variance schema onder Q_u moet er gelden dan $z(i) = 0 \quad \forall i$. Aangezien $z(i) \geq 0$, $A(i,j) \geq 0$ en $f(i) \geq 0$, betekent dat een noodzakelijke voorwaarde hiervoor is dat $f(i) = 0 \quad \forall i$. Stel we nemen de volgende uitdrukking voor Q_u:

$$Q(i,j) = \beta^{-\alpha} P(i,j) \frac{v(j)}{v(i)}$$

waarbij α nog vrij te kiezen is. Dit invullen in vergelijking (16) en gelijk aan 0 stellen geeft:

$$\begin{aligned}
\beta^{-2} \sum_{j=1}^{d} \frac{P^2(i,j)}{\beta^{-\alpha} P(i,j) \frac{v(j)}{v(i)}} \cdot v(j)^2 - v(i)^2 &= 0 \\
\beta^{-2+\alpha} \sum_{j=1}^{d} P(i,j) \frac{v(i)}{v(j)} \cdot v(j)^2 - v(i)^2 &= 0 \\
\beta^{-2+\alpha} \sum_{j=1}^{d} P(i,j) v(j) \cdot v(i) - v(i)^2 &= 0
\end{aligned}$$

Laat nu β een eigenwaarde zijn met bijbehorende eigenvector v van de matrix P. Dan geldt het volgende:

$$\begin{aligned}
\beta^{-2+\alpha} \sum_{j=1}^{d} P(i,j) v(j) \cdot v(i) - v(i)^2 &= 0 \\
\beta^{-2+\alpha} \Big(\beta \cdot v(i) \cdot v(i) \Big) - v(i)^2 &= 0 \\
\beta^{-1+\alpha} v(i)^2 - v(i)^2 &= 0
\end{aligned}$$

Dus de voorwaarde $f(i) = 0 \quad \forall i$ geeft dat $\alpha = 1$ en dat de zero-variance matrix Q de volgende uitdrukking heeft:

$$Q(i,j) = \beta^{-1} P(i,j) \frac{v(j)}{v(i)}$$

waarbij β een eigenwaarde moet zijn van de matrix P en v de bijbehorende eigenvector. In Desai's algoritme worden de dominante eigenwaarde en de eigenvector genoteerd met β_* en u^*. De zero-variance kansmaat Q_u^*, waarvoor dus geldt dat $z(i) = 0 \quad \forall i$, wordt gegeven door

$$Q_u^*(i,j) = \beta_*^{-1} P(i,j) \frac{u^*(j)}{u^*(i)}.$$

Onder deze kansmaat volstaat het dus om slechts één simulatie uit te voeren (voor iedere begintoestand $i \in S$). In sectie 3.2 is al aangegeven dat de zero-variance kansmaat in de praktijk niet te gebruiken is, aangezien de waarden

voor β_* en u^* juist de waarden zijn die bepaald moeten worden. Toch wordt deze kansmaat Q_u^* gebruikt om een geschikte kansmaat te vinden waaruit wel gesimuleerd kan worden. De kansmaat die in Desai's algoritme wordt toegepast, wordt gegeven door (voor $i, j \in S$):

$$Q_u(i,j) = \frac{P(i,j)u(j)}{\sum_{k \in S} P(i,k)u(k)}$$

In Desai's algoritme wordt de overgangsmatrix Q_u elke iteratie aangepast aan de huidige beste schatting van de eigenvector u^*. De gedachte hierachter is dat naarmate de schatting $\hat{u}$ van de eigenvector u^* beter wordt, de kansmaat $Q_{\hat{u}}$ dichter bij de zero-variance kansmaat komt te liggen:

$$Q_{\hat{u}}(i,j) = \frac{P(i,j)\hat{u}(j)}{\sum_{k \in S} P(i,k)\hat{u}(k)} \approx \frac{P(i,j)u^*(j)}{\beta_* u^*(i)} = \beta_*^{-1} P(i,j)\frac{u^*(j)}{u^*(i)}$$

Doordat de overgangsmatrix Q_u in elke iteratie van het algoritme wordt aangepast aan de huidige beste schatting, wordt het ook wel een adaptief of een lerend algoritme genoemd.

6.2 Het simuleren uit een discrete Markovketen

In Desai's algoritme wordt de Markovketen $Y = (Y_n : n \geq 0)$ gesimuleerd onder de overgangsmatrix Q_u. In deze sectie wordt uitgelegd hoe uit zo'n discrete Markovketen gesimuleerd kan worden aan de hand van de bijbehorende overgangsmatrix.
Laat Q_u de overgangsmatrix zijn waaruit de Markovketen gesimuleerd moet worden. Deze matrix heeft de volgende vorm:

$$Q_u = \begin{pmatrix} q_{11} & q_{12} & \dots & \dots & q_{1d} & 0 \\ q_{21} & q_{22} & \ddots & \ddots & & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ q_{d1} & & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \dots & 0 & 1 \end{pmatrix}$$

De overgangskansen worden dus gegeven door $\mathbb{P}(Y_{n+1} = j | Y_n = i) = q_{ij}$ Deze overgangsmatrix wordt getransformeerd tot een nieuwe matrix genoteerd met Q'_u. De kolommen van de matrix Q'_u wordt gevormd door de cumulatieve som te nemen van de kolommen van de matrix Q_u. Dit betekent het volgende: de eerste kolom van de matrix Q'_u is dezelfde kolom als van de originele matrix Q_u. Vervolgens is de tweede kolom van Q'_u gelijk aan de som van de eerste twee kolommen van Q_u. De derde kolom van Q'_u wordt gevormd door de eerste drie kolommen van Q_u bij elkaar op te tellen. In het algemeen geldt dus dat de $j-de$ kolom van Q'_u wordt gevormd door de eerste j kolommen van de matrix Q_u bij elkaar op te tellen. Aangezien de matrix Q_u stochastisch is, zal de laatste kolom van de nieuwe matrix Q'_u gevuld zijn met de waarde 1. Deze matrix ziet er dus als volgt uit

$$Q'_u = \begin{pmatrix} q'_{11} & q'_{12} & \dots & \dots & q'_{1d} & 1 \\ q'_{21} & q'_{22} & \ddots & \ddots & & 1 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ q'_{d1} & & \ddots & \ddots & \ddots & 1 \\ 0 & 0 & \dots & \dots & 0 & 1 \end{pmatrix}$$

waarbij $q'_{ij} = q_{i1} + q_{i2} + ... + q_{ij}$.
Laat de begintoestand gegeven zijn door $Y_0 = i$. Eerst wordt een trekking gedaan uit de uniforme verdeling op [0,1] en deze trekking wordt genoteerd met U (dus $U \sim U(0,1)$). Vervolgens wordt rij i van de matrix Q'_u geselecteerd en bepaald tussen welke twee waarden $q'_{i(j-1)}$ en $q'ij$ het getal U ligt. Aangezien $U \sim U(0,1)$, is de kans dat U tussen die twee waarden komt gelijk aan het verschil tussen die twee waarden:

$$\mathbb{P}(q'_{i(j-1)} < U < q'_{ij}) = q'_{ij} - q'_{i(j-1)} = q_{ij}$$

We zien dat de kans dat U tussen $q'_{i(j-1)}$ en q'_{ij} gelijk is aan de overgangskans van de Markovketen Y onder de originele matrix Q_u. Zodra U tussen $q'_{i(j-1)}$ en $q'ij$ nemen we dus voor de volgende toestand $Y_1 = j$. Dit proces herhaalt zich totdat de terugkeertoestand k wordt bereikt.

6.3 Convergentie naar u^*

Een belangrijk aspect van Desai's algoritme dat nog moet worden aangetoond, is of het algoritme ook daadwerkelijk convergeert naar de gevraagde eigenvector u^*. Aan het einde van sectie 5 is verteld dat de reeks van beste schattingen, die ontstaat bij het uitvoeren van het algoritme, kan worden opgevat als een Markovketen $X = (X_n : n \geq 0)$ met toestandsruimte $\mathbb{R}^d_+$. Desai heeft in zijn proefschrift (Desai, 2001) aangetoond met behulp van de theorie over Markovketens dat het algoritme een bijna zekere exponentiële convergentie heeft naar de eigenvector u^* zodra de steekproefgrootte n voldoende groot is. De stelling die Desai bewezen heeft, luidt als volgt:

Er bestaat een $N \in \mathbb{N}$ en $\rho \in (0,1)$ zodanig dat voor een steekproefgrootte van $n \geq N$ geldt:

$$\mathbb{P}\Big(||u^{(m)} - u^*|| \leq \rho^m \quad \text{voor alle behalve eindig veel m}\Big) = 1$$

waarbij $u^{(m)}$ de beste schatting is van iteratie m van het algoritme.

In deze sectie zal de exponentiële convergentie van Desai's algoritme voor een aantal matrices in de praktijk worden vastgesteld. Uit de stelling volgt dat deze convergentie afhangt van de steekproefgrootte die gebruikt wordt in het algoritme om tot een schatting van u^* te komen. We zullen daarom het algoritme meerdere malen op dezelfde matrix uitvoeren, alleen dan voor verschillende steekproefgrootten. Een belangrijk aspect is dat de exacte waarde u^* niet gebruikt zal worden om de nauwkeurigheid van de schatting $u^{(m)}$ te bepalen. In de praktijk zal de exacte waarde namelijk niet bekend zijn, aangezien het algoritme juist een benadering hiervan moet geven. De vraag is dan om toch een maat te vinden die een uitspraak kan doen over de grootte van $||u^{(m)} - u^*||$. De maat die nu geïntroduceerd wordt, is de zogeheten verwachte kwadratische fout (vanaf nu afgekort tot VKF) en is gedefinieerd als volgt:

$$VKF = \mathbb{E}\Big(||u^{(m)} - u^*||^2\Big)$$

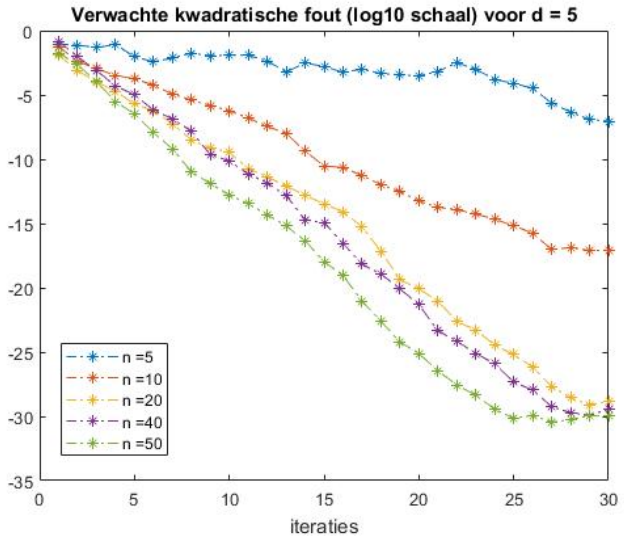
In eerste instantie lijkt het dat u^* nodig is om de waarde van de VKF te berekenen. Echter uit Desai's algoritme volgt dat $u^* = \mathbb{E}(u^{(m)})$ en deze eigenschap zal nu gebruikt worden om de VKF te herleiden tot de volgende vorm:

$$
\begin{aligned}
\mathbb{E}(||u^{(m)} - u^*||^2) &= \mathbb{E}\left(\sum_{i=1}^{d} \left(u_i^{(m)} - u_i^*\right)^2 \right) \\
&= \sum_{i=1}^{d} \mathbb{E}\left(\left(u_i^{(m)} - u_i^*\right)^2 \right) \\
&= \sum_{i=1}^{d} \mathbb{E}\left(\left(u_i^{(m)} - \mathbb{E}(u_i^{(m)})\right)^2 \right) \\
&= \sum_{i=1}^{d} Var\left(u_i^{(m)}\right) \\
&= \sum_{i=1}^{d} \frac{\sigma_i^2}{n} \\
&= \sum_{i=1}^{d} \left(\frac{\sigma_i}{\sqrt{n}}\right)^2
\end{aligned}
$$

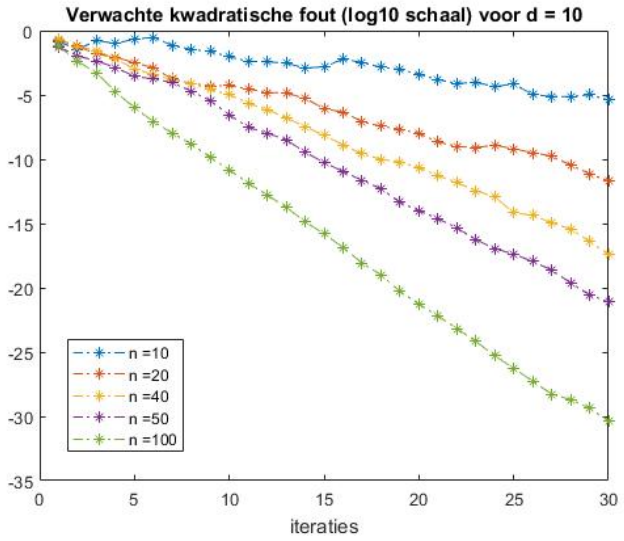
Hierbij is σ_i^2 de variantie van de n onafhankelijke simulaties van de schatting voor $u^*(i)$. We zien dus dat de VKF niets anders is dan de som van de standard errors in het kwadraat van de componenten van $u^{(m)}$. Bij het berekenen van de VKF zullen de varianties σ_i^2 worden vervangen door de steekproefvarianties $\hat{\sigma}_i^2$. We noteren de VKF dan ook vanaf nu met $V\hat{K}F$:

$$
V\hat{K}F = \sum_{i=1}^{d} \left(\frac{\hat{\sigma_i}}{\sqrt{n}}\right)^2
$$

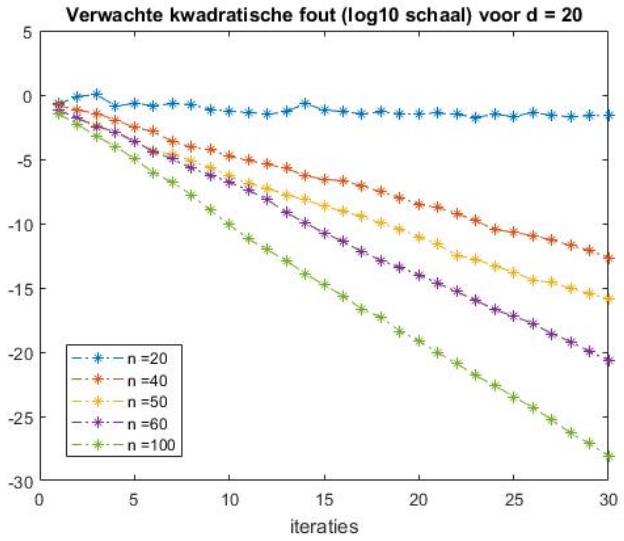
In figuur 1 t/m 3 is de $V\hat{K}F$ gegeven voor een matrix van dimensie 5, een matrix van dimensie 10 en een van dimensie 20. Hierbij is voor iedere matrix het algoritme uitgevoerd voor verschillende steekproefgrootten n. Hieruit is af te lezen dat de $V\hat{K}F$ exponentieel afneemt per iteratie zodra de steekproefgrootte voldoende groot is. Dit is in overeenstemming met de stelling aan het begin van deze sectie. Echter deze voldoende steekproefgrootte hangt af van de dimensie van de matrix. Zo is bij de matrix van dimensie 5 een steekproefgrootte van $n = 10$ al voldoende voor een exponentiële afname, terwijl bij de matrix van dimensie 20 een steekproefgrootte van $n = 20$ nog niet voldoende is om de $V\hat{K}F$ exponentieel af te laten nemen. Tevens geven de resultaten het vermoeden dat de snelheid van de convergentie (en dus de waarde van ρ) afhangt van de steekproefgrootte n en de dimensie van de matrix. In sectie 8 en 9 zal de invloed van deze en andere factoren op de convergentiesnelheid verder worden onderzocht.



Figuur 1: Verwachte kwadratische fout (op logaritmische schaal) voor een matrix van dimensie 5



Figuur 2: Verwachte kwadratische fout (op logaritmische schaal) voor een matrix van dimensie 10



Figuur 3: Verwachte kwadratische fout (op logaritmische schaal) voor een matrix van dimensie 20

7 De power methode

Nu de werking van Desai's algoritme is beschreven, willen we ook onderzoeken of dit algoritme een goede concurrent is voor de bestaande (deterministische) algoritmes. In dit onderzoek zal Desai's algoritme vergeleken worden met de power methode. De power methode is tevens een algoritme dat enkel de dominante eigenwaarde en eigenvector van een (willekeurige) matrix benaderd. In dit hoofdstuk zal de werking en de convergentiesnelheid van de power methode worden beschreven.

7.1 Werking power methode

Gegeven een matrix P waarvan de dominante eigenwaarde λ_1 en eigenvector u^* benaderd moeten worden. Laat u_0 een beginschatting zijn van de eigenvector. Dan convergeert het iteratieve proces

$$u_{m+1} = \frac{Pu_m}{||Pu_m||} = \frac{P^{m+1}u_0}{||P^{m+1}u_0||} \tag{18}$$

naar de eigenvector u^* van de matrix P die correspondeert met de dominante eigenwaarde λ_1, mits de algebraïsche en geometrische multipliciteit gelijk zijn en de andere eigenwaarden strikt kleiner (in absolute waarde) zijn dan λ_1. In elke iteratie wordt dus de vector u_m vermenigvuldigd met de matrix P en vervolgens genormaliseerd.
Verder kan bij iedere schatting u_m van de eigenvector de bijbehorende schatting van de eigenwaarde, genoteerd met θ_m, worden bepaald met de zogeheten Rayleigh quotiënt:

$$\theta_m = \frac{\langle u_m | Pu_m \rangle}{\langle u_m | u_m \rangle}.$$

Hierbij geldt dat $\theta_m \to \lambda_1$ voor $m \to \infty$.

7.2 Convergentie power methode

In deze sectie zal de convergentiesnelheid van de power methode worden vastgelegd. Om dit vast te leggen, zal een kort bewijs worden gegeven dat het iteratieve proces uit (18) inderdaad naar u^* convergeert. Het bewijs dat hier wordt gegeven, geldt onder de aanname dat de matrix P diagonaliseerbaar is. De matrices die in dit onderzoek beschouwd worden, zijn niet-negatief en irreducibel. De Perron-Frobenius stelling geeft ons dan dat de algebraïsche multipliciteit van de dominante eigenwaarden gelijk is aan 1. Dit houdt dus in dat $|\lambda_1| > |\lambda_i| \quad \forall i$ voor deze matrices.
Gegeven een $d \times d$ matrix P en laat $\lambda_1, \lambda_2, ..., \lambda_d$ de eigenwaarden van deze matrix zijn met bijbehorende eigenvectoren $v_1, v_2, ..., v_d$. Uit de aanname dat P diagonaliseerbaar is, volgt dat de matrix P kan worden getransformeerd tot de vorm

$$P = QDQ^{-1}$$

waarbij D een diagonaalmatrix is bestaande uit de eigenwaarden van de matrix P en de kolommen van Q bestaan uit de eigenvectoren van de matrix P. Verder is er een basis van eigenvectoren. Laat de beginschatting u_0 uitgedrukt zijn in deze basis van eigenvectoren:

$$u_0 = c_1 v_1 + c_2 v_2 + ... + c_d v_d$$

(waarbij $u_0 \neq 0$).
Dit resulteert in de volgende uitdrukking voor $P^m u_0$:

$$\begin{aligned} P^m u_0 &= P^m(c_1 v_1 + c_2 v_2 + ... + c_d v_d) \\ &= c_1 P^m v_1 + c_2 P^m v_2 + ... c_d P^m v_d \\ &= c_1 \lambda_1^m v_1 + c_2 \lambda_2^m v_2 + ... + c_d \lambda_d^m v_d \\ &= c_1 \lambda_1^m \left(v_1 + \frac{c_2}{c_1} \Big(\frac{\lambda_2}{\lambda_1} \Big)^m v_2 + ... + \frac{c_d}{c_1} \Big(\frac{\lambda_d}{\lambda_1} \Big)^m v_d \right) \end{aligned} \tag{19}$$

Aangezien λ_1 de unieke dominante eigenwaarde is van de matrix P, volgt dat

$$\left| \frac{\lambda_i}{\lambda_1} \right| < 1 \quad \forall i = 2, ..., d$$

en dus $\left| \frac{\lambda_i}{\lambda_1} \right|^m \to 0$ als $m \to \infty$. Dit betekent dus dat

$$P^m u_0 \to c_1 \lambda_1^m v_1$$

Door vervolgens te normaliseren convergeert u_m naar (een veelvoud van) de eigenvector behorende bij de dominante eigenwaarde λ_1:

$$u_{m+1} = \frac{P^m u_0}{||P^m u_0||} \to \frac{c_1 \lambda_1^m v_1}{||c_1 \lambda_1^m v_1||} = \frac{v_1}{||v_1||}$$

Uit vergelijking (19) is te zien dat de snelheid waarmee deze convergentie plaatsvindt, afhangt van de snelheid waarmee de termen $\left| \frac{\lambda_i}{\lambda_1} \right|^m$ naar 0 gaan. Aangezien $|\lambda_2| > |\lambda_3| > ... > |\lambda_d|$, zal de term $\left| \frac{\lambda_2}{\lambda_1} \right|^m$ het minst snel naar 0 toe convergeren. Dit betekent dus dat de convergentiesnelheid afhangt van de een-na-grootste eigenwaarde λ_2 en dat deze snelheid wordt gegeven door

$$||u^{(m)} - u^*|| \approx \mathcal{O}\left(\Big| \frac{\lambda_2}{\lambda_1} \Big|^m \right).$$

De power methode zal dus snel convergeren in het geval dat λ_2 (in absolute waarde) veel kleiner is dan λ_1 en deze snelheid neemt af naarmate het verschil tussen deze twee eigenwaarden kleiner wordt.

8 Complexiteitsanalyse

8.1 Wat is een complexiteitsanalyse?

Nu Desai's algoritme en de power methode besproken zijn, is het volgende doel om deze algoritmes met elkaar te gaan vergelijken om erachter te komen welk algoritme beter werkt voor het benaderen van de grootste eigenwaarde en bijbehorende eigenvector. De eerste vraag die beantwoord moet worden is dan: wat maakt een algoritme beter dan een ander algoritme? Om deze vraag te beantwoorden zijn twee eigenschappen van een algoritme van belang. Het eerste belangrijke aspect van een algoritme is de inspanning (ofwel de rekentijd) die nodig is om het algoritme uit te voeren. Hierbij wordt gekeken naar het aantal elementaire berekeningen die in een simulatie nodig zijn. De inspanning zal worden aangegeven in termen van zogeheten flops, waar een flop staat voor een elementaire berekening. Onder deze elementaire berekeningen vallen optellen, aftrekken, vermenigvuldigen, delen en machtsverheffing. Het tweede belangrijke aspect van een algoritme is de convergentiesnelheid naar de exacte oplossing. Stel je wilt maximaal $\epsilon > 0$ van de exacte oplossing af zitten. De convergentiesnelheid geeft dan een schatting hoeveel iteraties er ongeveer nodig zijn om tot deze nauwkeurigheid van ϵ te komen. Om algoritmes goed met elkaar te kunnen vergelijken, is het van belang zowel rekentijd als convergentiesnelheid van de algoritmes naast elkaar te leggen. Een algoritme dat bijvoorbeeld een zeer hoge convergentiesnelheid heeft, heeft misschien wel ook een zeer grote rekentijd nodig om tot deze convergentie te komen. Het is dan soms beter om een algoritme te kiezen waarbij de convergentiesnelheid wat lager ligt, maar dat de rekentijd ook aanzienlijk minder is. Daarom moeten deze twee aspecten, de rekentijd en de convergentiesnelheid, altijd samen in beschouwing worden genomen. Om deze twee aspecten samen te voegen, zullen we voor beide algoritmes een functie samenstellen die de hoeveelheid rekentijd aangeeft die nodig is om een zekere nauwkeurigheid ϵ te garanderen. Met het vergelijken van deze twee functies kunnen dan uitspraken worden gedaan over welk algoritme beter presteert. Om tot deze functies te komen, zal voor beide algoritmes eerst de totale inspanning worden vastgesteld, waarna vervolgens de convergentiesnelheid wordt bepaald. Het onderzoek doen naar de inspanning en convergentiesnelheid van een algoritme wordt ook wel een complexiteitsanalyse genoemd.

8.2 Complexiteitsanalyse Desai's algoritme

De complexiteitsanalyse van Desai's algoritme is beknopt beschreven in het proefschrift van Desai (Desai, 2001) en zal hier wat uitgebreider toegelicht worden.
Voor het bepalen van de totale inspanning van Desai's algoritme zal het algoritme worden opgesplitst in een aantal delen. Van elk deel van het algoritme wordt vervolgens de totale inspanning berekend. Aan het eind zullen deze inspanningen bij elkaar worden opgeteld om tot een totale inspanning van het algoritme te komen.

8.2.1 Inspanning berekenen van de matrix Q_u

Het eerste wat gebeurt in Desai's algoritme zodra een $d \times d$ matrix P wordt gegeven, is het berekenen van de nieuwe overgangsmatrix Q_u. We bepalen eerst het aantal berekeningen dat nodig is om één rij van de matrix Q_u te vormen. De elementen in de matrix Q_u worden bepaald door de volgende breuk te berekenen:

$$\frac{P(i,j)u(j)}{\sum_{k=1}^{d} P(i,k)u(k)}$$

Het berekenen van de som $\sum_{k=1}^{d} P(i,k)u(k)$ bevat d vermenigvuldigingen en $d-1$ optellingen en de nodige inspanning daarvoor is dus $\mathcal{O}(d)$. Voor het berekenen van alle factoren $P(i,j)u(j)$ van één rij zijn verder d vermenigvuldigingen nodig. Daarnaast zijn ook d delingen nodig om de breuken te berekenen en dus is de totale inspanning voor het berekenen van één rij $\mathcal{O}(d)$. Voor d rijen geldt dan dat de totale inspanning voor het berekenen van de matrix Q_u uitkomt op $\mathcal{O}(d^2)$.

8.2.2 Inspanning berekenen eigenwaarde λ

Zodra de matrix Q_u berekend is, wordt de Markovketen Y gesimuleerd onder Q_u totdat de terugkeertoestand k is bereikt. Na de simulatie wordt de volgende uitdrukking berekend:

$$W(i,\beta) = \beta^{-\tau} L_\tau \qquad\qquad Y_0 = i$$

waarbij L_τ wordt gegeven door vergelijking (13). Voor het berekenen van L_τ zijn in totaal τ delingen en τ vermenigvuldigingen nodig en de inspanning daarvoor is dus $\mathcal{O}(\tau)$. Voor het berekenen van $W(i,\beta)$ is vervolgens nog één machtsverheffing en één vermenigvuldiging nodig en dus is de totale inspanning om $W(i,\beta)$ te berekenen $\mathcal{O}(\tau)$. Een steekproefgrootte n betekent dat $W(i,\beta) = \beta^{-\tau} L_\tau$ n keer wordt gesimuleerd. Vervolgens wordt de gemiddelde functie berekend die wordt genoteerd als volgt

$$\tilde{W}(i,\beta) = \frac{1}{n} \sum_{l=1}^{n} \beta^{-\tau_l} L_{\tau_l}$$

en de inspanning voor het berekenen van $\tilde{W}(i,\beta)$ is dus $\mathcal{O}(n \cdot \tau)$. De schatting van de eigenwaarde, $\hat{\beta}$, wordt berekend door de volgende vergelijking op te lossen:

$$\tilde{W}(k,\hat{\beta}) = \frac{1}{n} \sum_{l=1}^{n} \hat{\beta}^{-\tau_l} L_{\tau_l} = 1$$

Aangezien het interval waar de exacte eigenwaarde zich bevindt berekend kan worden (zie sectie A.6 voor een bewijs hiervan), wordt de bisectiemethode gebruikt om voor deze vergelijking een oplossing te verkrijgen voor $\hat{\beta}$. Voor een nauwkeurigheid ϵ zijn $H = \log_2(\frac{b-a}{\epsilon})$ iteraties nodig, waarbij $[a,b]$ het interval is waar de exacte eigenwaarde λ zich bevindt. Dit betekent dus ook H

functie-evaluaties van de vorm $\tilde{W}(k, \hat{\beta})$. Alles bij elkaar genomen is de benodigde inspanning om een schatting voor de eigenwaarde λ te krijgen gelijk aan $\mathcal{O}(H \cdot n \cdot \tau)$

8.2.3 Inspanning berekenen eigenvector u

Zodra de schatting van de eigenwaarde berekend is, worden de componenten van de eigenvector berekend. De schattingen van de componenten worden genoteerd met $\hat{u}(j)$ voor $j = 1, 2, ..., d$. Uit het algoritme is te zien dat de componenten als volgt worden berekend:

$$\hat{u}(j) = \frac{1}{n} \sum_{l=1}^{n} \beta^{-\tau_l} L_{\tau_l}$$

Het berekenen van L_{τ_l} heeft weer een inspanning van $\mathcal{O}(\tau)$. Verder zijn voor het berekenen van $\hat{u}(j)$ n machtsverheffingen, n vermenigvuldigingen, n optellingen en één deling nodig. De totale inspanning om $\hat{u}(j)$ te bereken is daardoor $\mathcal{O}(n \cdot \tau)$. In totaal zijn er d componenten te berekenen, dus de inspanning voor het berekenen van de eigenvector is dan $\mathcal{O}(d \cdot n \cdot \tau)$

8.2.4 Inspanning berekenen random getallen

Tijdens het simuleren van de Markovketen Y onder de overgangsmatrix Q_u wordt gebruikt gemaakt van random getallen. Deze random getallen moeten gegenereerd worden en dit genereren heeft dan ook een bepaalde inspanning. We gaan ervan uit dat de inspanning van het genereren van één random getal hetzelfde is als die van één flop (dus $\mathcal{O}(1)$). Het simuleren van de Markovketen vanuit de begintoestand j geeft τ_j stappen door de keten, wat betekent dat er ook τ_j random getallen gegenereerd worden. Aangezien τ geen deterministische waarde is, zal voor het bepalen van het aantal operaties de verwachte waarde van τ gebruikt worden. Dus $\mathbb{E}(\tau_j)$ staat voor het verwachte aantal random getallen dat gegenereerd moet worden. Voor iedere begintoestand j worden n onafhankelijke simulaties uitgevoerd en dus zijn er $n \cdot \mathbb{E}(\tau_j)$ random getallen nodig voor iedere begintoestand. De totale hoeveelheid random getallen die nodig is voor één iteratie van het algoritme is $d \cdot n \cdot \mathbb{E}(\tau)$. Het genereren van één random getal heeft een inspanning van $\mathcal{O}(1)$, dus de totale inspanning voor het genereren van de random getallen is $\mathcal{O}(d \cdot n \cdot \mathbb{E}(\tau))$

8.2.5 Berekenen totale inspanning Desai's algoritme

Nu voor ieder deel van het algoritme de inspanning is bepaald, kan de totale inspanning van het algoritme worden vastgesteld. Een belangrijk punt is dat τ een stochastische variabele is. Dit betekent dat de totale inspanning van het algoritme niet deterministisch is. Om toch een uitspraak te kunnen doen over de totale inspanning, zal de zogeheten verwachte inspanning worden berekend. Dit wordt bereikt door τ te vervangen door $\mathbb{E}(\tau)$, het verwachte aantal operaties.

Het opsommen van de berekende inspanningen geeft dan

$$\mathcal{O}(d^2) + H \cdot \mathcal{O}(n \cdot \mathbb{E}(\tau)) + \mathcal{O}(d \cdot n \cdot \mathbb{E}(\tau)) + \mathcal{O}(d \cdot n \cdot \mathbb{E}(\tau)) = \mathcal{O}(d^2 + d \cdot n \cdot \mathbb{E}(\tau))$$

De verwachte totale inspanning voor één iteratie van het algoritme is dus

$$\mathcal{O}(d^2 + d \cdot n \cdot \mathbb{E}(\tau)).$$

8.2.6 Inspanning voor zekere nauwkeurigheid

In sectie 6.3 is verteld dat Desai's algoritme een bijna zekere exponentiële convergentie heeft zodra de steekproefgrootte n voldoende groot is, dus

$$\mathbb{P}\Big(||u^{(m)} - u^*|| \leq \rho^m \quad \text{voor alle behalve eindig veel m}\Big) = 1.$$

Voor een nauwkeurigheid van $\epsilon \geq 0$ (dus $||u^{(m)} - u^*|| \leq \epsilon$) geldt dat $\epsilon \approx \rho^m$. Het aantal iteraties m die nodig zijn om deze nauwkeurigheid te bereiken is dus

$$m \approx \log_\rho(\epsilon)$$

Voor een nauwkeurigheid van ϵ zijn dus ongeveer $\log_\rho(\epsilon)$ iteraties van het algoritme nodig.

Dus de verwachte totale inspanning die nodig is om tot een nauwkeurigheid van ϵ te komen is $\mathcal{O}\Big(\big(d^2 + d \cdot n \cdot \mathbb{E}(\tau)\big) \cdot \log_\rho(\epsilon)\Big)$

8.3 Complexiteitsanalyse power methode

Om Desai's algoritme te kunnen vergelijken met de power methode, zal ook van dit algoritme de totale inspanning bepaald worden die nodig is om een nauwkeurigheid ϵ te bereiken.

8.3.1 Totale inspanning bij power methode

Iedere iteratie van de power methode bestaat uit het berekenen van een matrix-vector product. Laat P een $d \times d$ matrix zijn en u_m de schatting van de eigenvector na m iteraties. Voor het berekenen van iedere component van Pu_m zijn d vermenigvuldigingen en $d-1$ optellingen nodig, wat een totale inspanning van $\mathcal{O}(d)$ is. Totaal zijn er d componenten, dus de totale inspanning om het product Pu_m te berekenen is $\mathcal{O}(d^2)$.

8.3.2 Inspanning voor zekere nauwkeurigheid

In sectie 7.2 is aangetoond dat de convergentie van de power methode exponentieel is met ratio $\frac{\lambda_2}{\lambda_1}$. Dit houdt in dat het verschil tussen de schatting en de exacte waarde van de eigenvector wordt gegeven door

$$||u^{(m)} - u^*|| \approx \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^m\right)$$

Om tot een nauwkeurigheid van ϵ te komen, geldt dat

$$\begin{aligned} \left|\frac{\lambda_2}{\lambda_1}\right|^m &\approx \epsilon \\ m &\approx \log_{\frac{\lambda_2}{\lambda_1}}(\epsilon) \end{aligned}$$

Dus er zijn ongeveer $\log_{\frac{\lambda_2}{\lambda_1}}(\epsilon)$ iteraties nodig om tot een nauwkeurigheid van ϵ te komen. De totale inspanning die nodig is om tot een nauwkeurigheid van ϵ te komen is $\mathcal{O}\Big(d^2 \cdot \log_{\frac{\lambda_2}{\lambda_1}}(\epsilon)\Big)$.

8.4 Vergelijken Desai's algoritme en power methode

Nu voor zowel Desai's algoritme als de power methode een complexitetisanalyse is gemaakt, zullen we deze kort met elkaar vergelijken. De totale inspanning voor een zeker nauwkeurigheid ϵ is dus voor beide algoritmes als volgt:

$$\begin{aligned} \text{Desai's algoritme} &: \mathcal{O}\Big(\big(d^2 + d \cdot n \cdot \mathbb{E}(\tau)\big) \cdot \log_\rho(\epsilon)\Big) \\ \text{power methode} &: \mathcal{O}\Big(d^2 \cdot \log_{\frac{\lambda_2}{\lambda_1}}(\epsilon)\Big) \end{aligned}$$

Een belangrijke opmerking is dat voor Desai's algoritme een verwachte inspanning is berekend. Uit de complexiteitsanalyses is te zien dat voor beide algoritmes de dimensie van invloed is op de inspanning. Ook is af te lezen dat de inspanning van de power methode in alle gevallen kleiner of gelijk is aan die van Desai's algoritme (dit hangt af van de waarde van $\mathbb{E}(\tau)$). De convergentiefactoren ρ en $\frac{\lambda_2}{\lambda_1}$ zijn moeilijk met elkaar te vergelijken, aangezien de waarde van ρ in Desai's algoritme onbekend is. Tevens is niet bekend welke factoren van invloed zijn op de waarde van ρ.
In de volgende sectie zullen we Desai's algoritme en de power methode in de praktijk met elkaar vergelijken. Hierbij zal ook voor een aantal factoren bepaald worden of deze van invloed zijn op de convergentiefactor ρ in Desai's algoritme.

9 Enkele simulaties: Desai's algoritme vs power methode

Nu de (theoretische) complexiteitsanalyses van Desai's algoritme en de power methode zijn beschreven, is de volgende stap om de convergentie van de algoritmes in de praktijk te gaan onderzoeken. Naast het vergelijken van de algoritmes, zal ook worden gekeken naar een aantal factoren die mogelijk van invloed zijn op de convergentiesnelheid van Desai's algoritme. Hiervoor zullen een aantal simulaties gedaan worden. Uit de complexiteitsanalyses zijn al een aantal eigenschappen naar voren gekomen die invloed hebben op de convergentiesnelheid van de algoritmes. Zo spelen de dimensie van de matrix en de een-na-grootste eigenwaarde een belangrijke rol. In dit onderzoek zullen de volgende eigenschappen die (mogelijk) invloed hebben op de convergentiesnelheid worden onderzocht:

1. De dimensie van de matrix
2. De keuze voor de terugkeertoestand k in Desai's algoritme
3. De een-na-grootste eigenwaarde van de matrix
4. De beginschatting u^0 van de eigenvector u^*

Om de invloed van deze eigenschappen te onderzoeken op de convergentiesnelheid, zullen een aantal matrices geselecteerd worden waarop beide algoritmes worden toegepast. De keuze voor deze testmatrices zal verder worden toegelicht in sectie 9.2.
Zodra deze testmatrices geselecteerd zijn, worden zowel Desai's algoritme als de power methode hierop uitgevoerd. Belangrijk is dat zodra een eigenschap wordt onderzocht, bijvoorbeeld de invloed van de een-na-grootste eigenwaarde, de andere factoren die mogelijk van invloed zijn op de convergentie gelijk blijven. Tenzij anders vermeld, zullen de volgende kenmerken altijd worden toegepast.

- 20 iteraties van beide algoritmes worden uitgevoerd
- Desai's algoritme wordt voor iedere matrix 10x uitgevoerd
- De vectoren u^0, u^{20} en u^* zijn genormaliseerd zodat $||u^0|| = ||u^{20}|| = ||u^*|| = 1$

Het meerdere malen uitvoeren van Desai's algoritme wordt gedaan om ook de invloed van de 'randomness' te beperken. Vervolgens wordt van iedere uitvoering van Desai's algoritme de beginfout $||u^0 - u^*||$ en de uiteindelijke fout $||u^{20} - u^*||$ berekend. Hierbij wordt u^* bepaald met de ingebouwde functie *eig()* in MATLAB. Daarna wordt de gemiddelde fout van de 10 simulaties als volgt bepaald:

$$\text{gemiddelde fout} = \frac{1}{10}\sum_{i=1}^{10}\frac{||u_i^{20} - u^*||}{||u_i^0 - u^*||}$$

Naast de gemiddelde fout zal ook de kleinste fout en de grootste fout worden gegeven om de totale range van de convergentiesnelheden aan te geven. Indien deze fouten in dezelfde orde van grootte zitten, zal de range niet worden weergegeven
Voor het bepalen van de convergentiesnelheid is alleen de orde van grootte van de fout belangrijk. Tevens zal de fout op logaritmische schaal worden genoteerd en worden afgerond op een geheel getal.
Om een uitspraak te doen over de convergentiesnelheid, zal ook de convergentiefactor ρ worden bepaald. Belangrijk om rekening mee te houden is de zogeheten machine precisie van MATLAB die voor een limiet zorgt op de maximale nauwkeurigheid voor de schatting van u^*. Op basis van een aantal resultaten met Desai's algoritme is te zien dat deze machine precisie ligt rond een waarde van 10^{-15}. Zodra de fout al voor de 20 iteraties deze waarde heeft bereikt, zal de convergentiefactor ρ als volgt worden bepaald. Laat m de eerste iteratie zijn waarbij de fout de machine precisie heeft berreikt. Dan wordt ρ als volgt berekend:

$$\rho = \left(\frac{||u^{(m)} - u^*||}{||u^{(0)} - u^*||} \right)^{\frac{1}{m}}$$

De term $||u^{(0)} - u^*||$ zorgt ervoor dat de beginfout wordt meegewogen in de uiteindelijke convergentiesnelheid. Indien na 20 iteraties de fout nog groter is dan 10^{-15}, zal ρ bepaald worden door $m = 20$ in bovenstaande uitdrukking in te vullen.
Een andere eigenschap van Desai's algoritme dat eventueel invloed kan hebben op de convergentie is de steekproefgrootte n. We zullen in de volgende sectie vaststellen welke steekproefgrootte geschikt is om het algoritme mee uit te voeren. Zodra deze steekproefgrootte bepaald is, zullen alle simulaties hiermee uitgevoerd worden.

9.1 De keuze voor de steekproefgrootte

In Desai's algoritme is de steekproefgrootte van belang. Om te bepalen of een grotere steekproefgrootte opweegt tegen de toename in de rekentijd van het algoritme, zullen we het algoritme laten werken op 50 matrices van dimensie 5 die random worden gegenereerd. Het random genereren van de matrices in MATLAB verzekert ons ook dat de matrices niet-negatief en irreducibel zijn. Ieder element van de matrix wordt dan bepaald aan de hand van een trekking uit de $U(0, 1)$ verdeling. Er wordt een zogenaamde 'random seed' in MATLAB gebruikt tijdens het genereren van de matrices, waardoor de resultaten reproduceerbaar zijn. Iedere matrix wordt vervolgens met verschillende steekproefgrootte doorlopen, van $n = 25$ oplopend tot $n = 200$. De beginschatting is voor iedere matrix gelijk aan $u^0 = [1/\sqrt{5}, 1/\sqrt{5}, ..., 1/\sqrt{5}]^T$ (merk op dat dan geldt dat $||u^0|| = 1$). Vervolgens wordt de fout

$$\frac{||u^{(20)} - u^*||}{||u^{(0)} - u^*||}$$

na 20 iteraties berekend. Tevens zal berekend worden hoeveel iteraties er gemiddeld nodig zijn om de fout met een factor 10 te verkleinen (en dus een decimaal zekerheid vast te stellen van de exacte waarde u^*). Als laatste zal ook berekend worden hoeveel seconden het kost om de fout een factor 10 te verkleinen. We berekenen dus zowel een maat voor de nauwkeurigheid als een maat voor de rekentijd. De resultaten van de 50 matrices zijn gegeven in tabel 1.

Steekproefgrootte	Fout en range	Iteraties/decimaal	Seconden/decimaal
n=25	-7 [-14, -7]	2.9	0.035
n=50	-9 [-15, -8]	2.2	0.049
n=100	-10[-15, -8]	2.0	0.080
n=200	-15 [-15, -14]	1.3	0.16

Tabel 1: De gemiddelde fout (op logaritmische schaal) van 50 random matrices na 20 iteraties van Desai's algoritme voor verschillende steekproefgrootten. Tevens is het aantal iteraties en het aantal seconden gegeven dat nodig is om de fout met een factor 10 te verkleinen.

In de tabel is te zien dat voor een verdubbeling van de steekproefgrootte het aantal iteraties dat nodig is om de fout met een factor 10 te reduceren afneemt, terwijl de rekentijd die daarvoor nodig is toeneemt. Echter, de rekentijd neemt sneller toe dan dat het aantal iteraties afneemt. Voor een verandering van de steekproefgrootte van 25 naar 50 neemt het aantal iteraties dat nodig is voor een decimaal nauwkeurigheid af met 24%, terwijl de toename in rekentijd daarvoor 40% bedraagt.Het is dus niet gunstiger om met een grotere steekproefgrootte te gaan simuleren, maar met een kleinere steekproefgrootte bijvoorbeeld een aantal iteraties meer te berekenen. Wel is het van belang dat de steekproefgrootte groot genoeg is om een exponentiële convergentie te krijgen (zie de stelling uit sectie 6.3). Voor matrices van dimensie 20 was de steekproefgrootte van $n = 25$ niet groot genoeg om convergentie te laten plaatsvinden. Om ook de convergentie te garanderen bij matrices van dimensie 20, zullen we bij de testmatrices dan ook een steekproefgrootte van $n = 50$ hanteren.
Om een eerste indruk te krijgen of Desai's algoritme een goed alternatief is voor de power methode, is op de 50 matrices ook de power methode toegepast. De resultaten zijn als volgt:

	Fout en range	Iteraties/decimaal	seconden/decimaal
power methode	-9 [-16, -7]	2.2	7.3e-04

Tabel 2: De gemiddelde fout (op logaritmische schaal) van 50 random matrices na 20 iteraties van de power methode voor verschillende steekproefgrootten. Tevens is het aantal iteraties en het aantal seconden gegeven dat nodig is om de fout met een factor 10 te verkleinen.

Hieruit is te zien dat de convergentie vergelijkbaar is met het Desai algoritme bij een steekproefgrootte van $n = 50$. Echter het aantal seconden dat nodig is om tot deze nauwkeurigheid te komen is vele malen kleiner dan bij Desai's algoritme. Om te kijken of dit grote tijdverschil nog afhankelijk is van de dimensie, kan een beroep gedaan worden op de complexiteitsanalyses van beide algoritmes uit hoofdstuk 8. Deze complexiteitsanalyses laten zien dat voor een toename in dimensie de rekentijd voor beide algoritmes ongeveer kwadratisch toeneemt. Hieruit kan geconcludeerd worden dat ook voor hogere dimensies de rekentijd van de power methode veel minder zal zijn dan die van Desai's algoritme. Gebaseerd op deze 50 matrices (en de complexiteitsanalyse) is de power methode dus de favoriet boven Desai's algoritme.

9.2 De testmatrices

Nu de steekproefgrootte voor het simuleren is vastgesteld, is de volgende stap om een aantal matrices te selecteren waarop Desai's algoritme en de power methode worden toegepast. De matrices moeten niet-negatief en irreducibel zijn en kunnen dus niet zomaar willekeurig worden gekozen. Een manier om matrices te verkrijgen met deze eigenschappen is het genereren van random matrices zoals is uitgelegd in de vorige sectie. Het nadeel van het random genereren van matrices is dat eigenschappen die onderzocht willen worden, zoals de invloed van de een-na-grootste eigenwaarde, niet expliciet uitgelicht kunnen worden. Het toeval bepaalt wat de tweede grootste eigenwaarde van de matrix zal zijn. Daarom zal een andere methode moeten worden toegepast om geschikte testmatrices te selecteren. Hierbij kunnen matrices worden gevormd aan de hand van de gevraagde eigenwaarden die deze matrix moet bevatten. Het construeren van deze matrices is gebaseerd op het algoritme uit het artikel van Lin (2015). Dit algoritme genereert symmetrische, niet-negatieve matrices aan de hand van de gevraagde (reële) eigenwaarden die de matrix moet hebben. Aangezien de invloed van de een-na-grootste eigenwaarde wordt onderzocht, kunnen met behulp van dit algoritme matrices specifiek gekozen worden op basis van deze tweede eigenwaarde. De eis dat de matrices irreducibel moeten zijn, zorgt ervoor dat alleen de dominante eigenwaarde positief mag zijn. Een andere voorwaarde van het algoritme is dat de som van alle eigenwaarden niet negatief mag zijn. Zodra aan deze voorwaarden wordt voldaan, genereert het algoritme niet-negatieve, irreducibele matrices waarmee de simulaties uitgevoerd kunnen worden. Verder zal de keuze van de matrices gebaseerd zijn op de ratio's $|\lambda_2/\lambda_1|$. Aan de hand van deze voorwaarden zullen de matrices worden geselecteerd. Voor dimensie 5 worden vier testmatrices zo geconstrueerd dat de ratio's $|\lambda_2/\lambda_1|$ oplopen van 0.2 voor matrix 1 tot 0.8 voor matrix 4. Ook voor dimensie 10 en 20 zijn de matrices zo gekozen dat de ratio's $|\lambda_2/\lambda_1|$ van elkaar verschillen (0.2 om 0.6). De testmatrices die gebruikt worden voor de simulaties bezitten de volgende eigenwaarden:

Dimensie 5
Matrix 1: eigenwaarden (20, -1 (3x), -4)
Matrix 2: eigenwaarden (20, -1 (3x), -8)

Matrix 3: eigenwaarden (20, -1 (3x), -12)
Matrix 4: eigenwaarden (20, -1 (3x), -16)

Dimensie 10
Matrix 5: eigenwaarden (20, -1 (2x), -2 (4x), -3 (2x), -4)
Matrix 6: eigenwaarden (50, -1 (2x), -2 (4x), -3 (2x), -30)

Dimensie 20
Matrix 7: eigenwaarden (20, -$\frac{1}{2}$ (4x), -1 (14x), -4)
Matrix 8: eigenwaarden (50, -1 (18x), -30)

Indien de eigenwaarden van een matrix met eenzelfde factor worden vermenigvuldigd, heeft dit geen effect op de convergentiesnelheid. Aan de hand van het algoritme uit het artikel van Lin (2015) worden de matrices dus gegenereerd. Voor matrix 1 geeft dat, na genormaliseerd te zijn (zoals beschreven in sectie 4), het volgende resultaat:

$$\begin{pmatrix} 0 & 0.0306 & 0.0306 & 0.0566 & 0.1167 \\ 0.0306 & 0 & 0.0306 & 0.0566 & 0.1167 \\ 0.0306 & 0.0306 & 0 & 0.0566 & 0.1167 \\ 0.0566 & 0.0566 & 0.0566 & 0 & 0.2522 \\ 0.1167 & 0.1167 & 0.1167 & 0.2522 & 0.3976 \end{pmatrix}$$

Voor de andere matrices kan op eenzelfde manier een uitdrukking worden gevonden.
Aan de hand van deze acht testmatrices zal nu de invloed van de vier factoren (zie bladzijde 34) op Desai's algoritme worden bepaald.

9.3 De terugkeertoestand k in Desai's algoritme

In Desai's algoritme wordt een terugkeertoestand k van de te simuleren matrix Q_u gekozen. Zodra deze toestand bereikt wordt, stopt de simulatie en wordt de functie $W(i, \beta)$ berekend. De vraag is dan welke terugkeertoestanden moeten worden geselecteerd om simulaties mee uit te voeren. In dit onderzoek selecteren we van iedere matrix 3 terugkeertoestanden. Deze 3 terugkeertoestanden zijn gebaseerd op de kolomsommen van de matrix. Nu zal voor een grotere kolomsom de terugkeerkans naar die toestand groter zijn dan bij een de kleine kolomsom. We selecteren dus de terugkeertoestanden die horen bij de kleinste kolomsom, de middelste kolomsom en de grootste kolomsom. Voor een aantal van de testmatrices van dimensie 5 en 10 zijn de resultaten weergegeven in tabel 3.

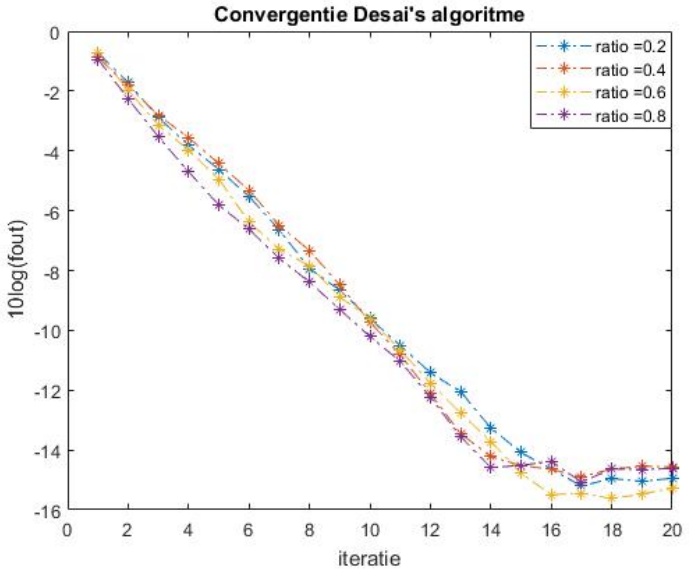
	kleinste som		middelste som		grootste som	
Dimensie 5						
Matrix 1	-6	[-11, -6]	-14	[-16, -14]	-15	[-16, -15]
Matrix 2	-7	[-10, -6]	-15		-15	
Matrix 3	-8	[-10, -7]	-15	[-16, -15]	-16	[-16, -15]
Dimensie 10						
Matrix 5	-4	[-6, -4]	-8	[-9, -8]	-11	[-12, -11]
Matrix 6	-3	[-6, -2]	-7	[-8, -6]	-15	

Tabel 3: Gemiddelde fout (op log 10 schaal) en range na 20 iteraties voor de verschillende terugkeertoestanden behorende bij de kleinste, middelste en grootste kolomsom

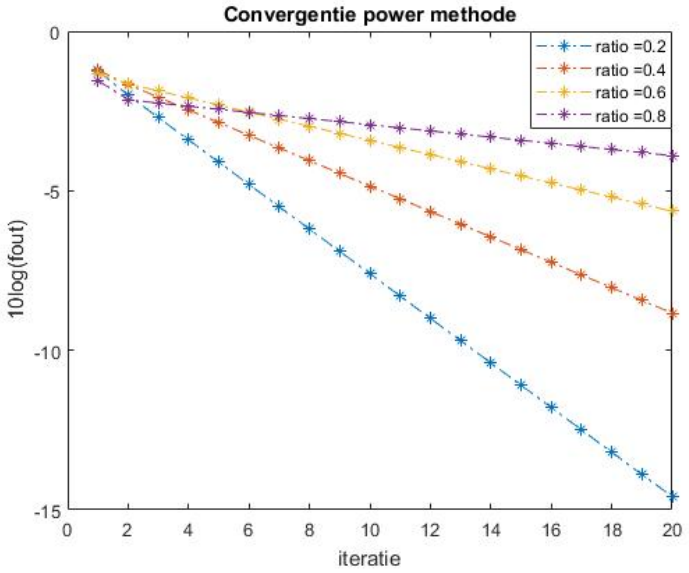
Uit de tabel is te zien dat convergentiesnelheid het grootst is voor de terugkeertoestand behorende bij de grootste kolomsom van de matrix. Dit is zowel het geval bij de matrices van dimensie 5 als die van dimensie 10. Tevens is de totale range van de fouten ook veel kleiner naarmate de kolomsom toeneemt. Gebaseerd op deze resultaten zullen we voor de volgende simulaties altijd de terugkeertoestand gebruiken behorende bij de grootste kolomsom van de matrix.

9.4 De een-na-grootste eigenwaarde van de matrix

Van de power methode is bekend dat de een-na-grootste eigenwaarde van de matrix bepalend is voor de convergentiesnelheid van het algoritme. De vraag is of voor Desai's algoritme deze factor ook invloed heeft op de convergentiesnelheid. De testmatrices uit sectie 9.2 zijn zo gekozen dat de ratio's $|\lambda_2/\lambda_1|$ goed van elkaar verschillen. Voor Desai's algoritme zal weer de gemiddelde fout worden gegeven op basis van de 10 verschillende simulaties die worden uitgevoerd op dezelfde matrix. Het eerste wat nog bevestigd moet worden, is of de convergentie van beide algoritmes ook daadwerkelijk exponentieel verloopt. Om dit te kunnen aantonen, is de convergentie van de testmatrices van dimensie 5 grafisch weergegeven in de figuren 4 en 5. Hierbij is het verloop van de fout (op logaritmische schaal) uitgezet tegen het aantal iteraties dat in het algoritme is doorlopen. In deze figuren is goed te zien dat de convergentie voor beide algoritmes inderdaad exponentieel is.



Figuur 4: Fout (op logaritmische schaal) van Desai's algoritme voor diverse ratio's $|\lambda_2/\lambda_1|$ bij matrices van dimensie 5



Figuur 5: Fout (op logaritmische schaal) van de power methode voor diverse ratio's $|\lambda_2/\lambda_1|$ bij matrices van dimensie 5

Voor de vier verschillende matrices van dimensie 5 zijn de resultaten voor Desai's algoritme weergegeven in tabel 4:

Matrix	Ratio	Fout	ρ	Seconden/decimaal
1	0.2	-15	0.10	0.036
2	0.4	-15	0.11	0.036
2	0.6	-15	0.10	0.034
4	0.8	-15	0.11	0.037

Tabel 4: Convergentie Desai's algoritme voor diverse ratio's $|\lambda_2/\lambda_1|$ van matrices met dimensie 5. Hierbij is de gemiddelde fout gegeven op een log 10 schaal na 20 iteraties. Tevens is ook de convergentiefactor ρ bepaald en de tijd die nodig is om de fout met een factor 10 te verkleinen.

Hieruit is af te lezen dat de ratio $|\lambda_2/\lambda_1|$ geen invloed heeft op de convergentiesnelheid van het algoritme. Ook in figuur 4 is goed te zien dat voor iedere matrix de convergentie op eenzelfde manier verloopt. Ondanks de toename van de ratio's (en dus de een-na-grootste eigenwaarde dichter bij de dominante eigenwaarde komt te liggen) loopt het algoritme na ongeveer 15 iteraties tegen de machine precisie aan. Ook de waardes voor ρ zijn hierdoor nagenoeg gelijk voor de verschillende matrices. Op basis van deze testmatrices ontstaat het vermoeden dat de een-na-grootste eignwaarde geen invloed heeft op de convergentiesnelheid voor Desai's algoritme.
Voor de power methode zou de convergentie duidelijk moeten afnemen naarmate de ratio $|\lambda_2/\lambda_1|$ toeneemt. In tabel 5 is te zien dat dit inderdaad het geval is. Ook in figuur 5 is de invloed van de ratio op de convergentiesnelheid goed te zien. Verder is uit de laatste kolommen van de tabellen 4 en 5 af te lezen dat de power methode, ondanks de invloed van de een-na-grootste eigenwaarde, vele malen sneller convergeert dan Desai's algoritme.

Matrix	Ratio	Fout	ρ	Seconden/decimaal
1	0.2	-15	0.19	7.3e-04
2	0.4	-9	0.37	8.0e-04
2	0.6	-6	0.54	1.3e-03
4	0.8	-4	0.66	2.0e-03

Tabel 5: Convergentie power methode voor diverse ratio's $|\lambda_2/\lambda_1|$ van matrices met dimensie 5. Hierbij is de gemiddelde fout gegeven op een log 10 schaal. Tevens is ook de convergentiefactor ρ gegeven.

Om ook de invloed van de dimensie mee te nemen, is ook voor een tweetal matrices van dimensie 10 en dimensie 20 onderzocht op de convergentiesnelheid voor twee verschillende ratio's $|\lambda_2/\lambda_1|$. De resultaten hiervan zijn te vinden in tabel 6 en 7.

	ratio	fout		ρ
Dimensie 10				
Matrix 5	0.2	-11	[-12, -11]	0.33
Matrix 6	0.6	-15		0.14
Dimensie 20				
Matrix 7	0.2	-9	[-10, -9]	0.35
Matrix 8	0.6	-15		0.18

Tabel 6: Convergentie Desai's algoritme voor verschillende ratio's $|\lambda_2/\lambda_1|$ van matrices met dimensie 10 en 20. Hierbij is de gemiddelde fout gegeven op een log 10 schaal. Tevens is ook de convergentiefactor ρ gegeven

	ratio	fout	ρ
Dimensie 10			
Matrix 5	0.2	-15	0.18
Matrix 6	0.6	-7	0.5
Dimensie 20			
Matrix 7	0.2	-15	0.18
Matrix 8	0.6	-7	0.46

Tabel 7: Convergentie power methode voor verschillende ratio's $|\lambda_2/\lambda_1|$ van matrices met dimensie 10 en 20. Hierbij is de gemiddelde fout gegeven op een log 10 schaal. Tevens is ook de convergentiefactor ρ gegeven

Bij Desai's algoritme lijkt het erop dat de convergentiesnelheid toeneemt naarmate de ratio ook toeneemt. Echter, of de invloed van de een-na-grootste eigenwaarde hier een rol speelt, is moeilijk te zeggen. In tabel 3 is namelijk te zien dat voor testmatrix 6 de gemiddelde fout een factor 10^8 (!) kan verschilen zodra de terugkeertoestanden anders zijn. Het lijkt er dus op dat vooral de terugkeertoestand een belangrijke factor speelt voor de convergentiesnelheid. Voor de power methode is weer te zien dat de convergentiesnelheid afneemt naarmate de ratio toeneemt.

9.5 De beginschatting u^0 van de eigenvector u^*

In zowel Desai's algoritme als de power methode moet een beginschatting u^0 van de eigenvector u^* worden gekozen. De beginschatting wordt genormaliseerd en dus zal de lengte van de beginschatting u^0 geen invloed op de convergentiesnelheid. Een maat om te bepalen of een beginschatting u^0 in de buurt van de exacte waarde ligt, is de hoek tussen de vector u_0 en de vector u^*. Naarmate de hoek groter wordt, zal de beginschatting verder van de exacte waarde af komen te liggen. De vraag is dan of dit ook grote invloed heeft op de uiteindelijke convergentiesnelheid. Doordat de vectoren u^0 en u^* beide genormaliseerd worden, verschilt alleen de hoek van de vectoren. Voor het bepalen van de hoek θ tussen de vectoren u^0 en u^* geldt de volgende relatie :

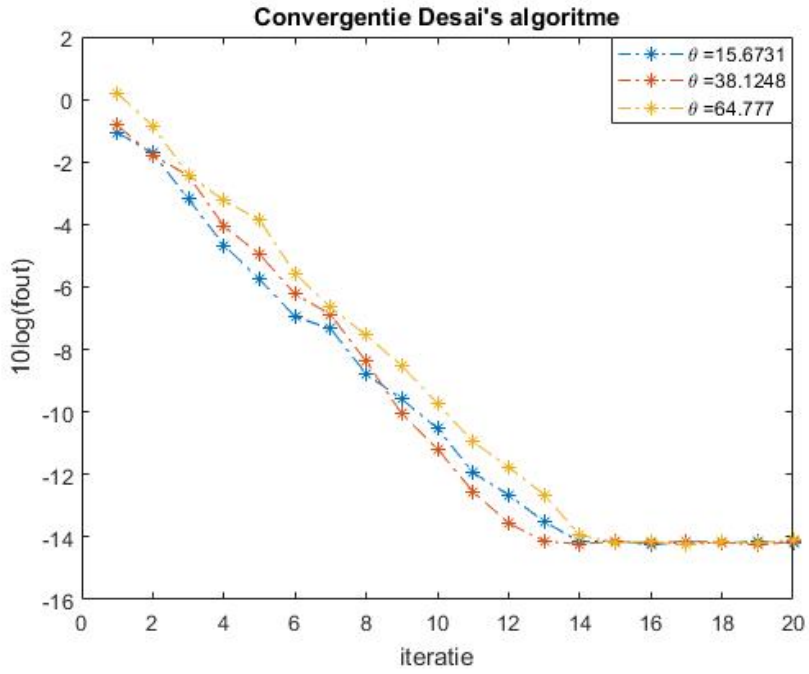
$$\theta = \cos^{-1}\left(\frac{u^0 \cdot u^*}{||u^0|| \cdot ||u^*||}\right) = \cos^{-1}(u^0 \cdot u^*)$$

waarbij $u^0 \cdot u^*$ het inwendig product is van de vectoren u^0 en u^*. Hieronder wordt de hoek θ in graden weergegeven. Voor het onderzoek zal de testmatrix 1 uit sectie 9.2 worden gebruikt. Voor matrix 1 zijn de resultaten van Desai's algoritme gegeven in tabel 8.

θ	Fout	ρ
16°	-15	0.09
38°	-15	0.09
65°	-15	0.09

Tabel 8: Gemiddelde fout Desai's algoritme bij matrix 1 voor verschillende beginschattingen u^0. Ook de convergentiefactor ρ is gegeven.

Uit de tabel is zien dat voor de verschillende hoeken θ de convergentiesnelheden exact hetzelfde zijn. Na ongeveer 14 iteraties loopt de grootte van de fout tegen de machine precisie aan. Dit is ook grafisch te zien in figuur 6.



Figuur 6: Fout (op logaritmische schaal) van Desai's algoritme voor matrix 1 voor diverse beginschattingen u^0

In tabel 9 is te zien dat ook voor de power methode de beginschatting geen grote invloed heeft op de convergentiesnelheid.

θ	Fout	ρ
16°	-15	0.19
38°	-15	0.19
65°	-15	0.19

Tabel 9: Gemiddelde fout power methode bij matrix 1 voor verschillende beginschattingen u^0. Ook de convergentiefactor ρ is gegeven.

Om ook de invloed van de dimensie in beschouwing te nemen, zal ook op testmatrix 5 (met een dimensie van 10) Desai's algoritme voor verschillende beginschattingen u^0 worden uitgevoerd. De resultaten hiervan zijn te vinden in tabel 10.

θ	Fout		ρ
16°	-10	[-11, -10]	0.22
24°	-10	[-11, -10]	0.20
34°	-10	[-11, 10]	0.21

Tabel 10: Gemiddelde fout Desai's algoritme bij matrix 5 voor verschillende beginschattingen u^0. Ook de convergentiefactor ρ is gegeven.

Ook deze resultaten laten zien dat de beginschatting nauwelijks invloed heeft op de convergentiesnelheid. Om te laten zien dat de keuze voor de terugkeertoestand veel bepalender is voor de convergentiesnelheid, zal opnieuw op matrix 5 Desai's algoritme worden toegepast. Het enige verschil is nu dat de terugkeertoestand wordt gekozen die behoort bij de middelste kolomsom. De resultaten hiervan zijn te vinden in tabel 11.

θ	Fout		ρ
16°	-8	[-9, -8]	0.30
24°	-7	[-8,-7]	0.35
34°	-6	[-8,-6]	0.37

Tabel 11: Gemiddelde fout Desai's algoritme bij matrix 5 voor verschillende beginschattingen u^0. Hierbij is de terugkeertoestand gekozen behorende bij de middelste kolomsom. Ook de convergentiefactor ρ is gegeven.

Vergelijken we tabel 10 en 11 met elkaar, dan zien we dat de convergentiesnelheid fors lager is zodra een andere terugkeertoestand wordt gekozen. Zo is bij een beginfout van 34° de gemiddelde fout uiteindelijk een factor 10^4 groter bij een 'slechtere' terugkeertoestand. Net als bij de een-na-grootste eigenwaarde is de conclusie dat de beginschatting niet van grote invloed is op de convergentiesnelheid. De keuze van de terugkeertoestand daarentegen is veel meer bepalend.

10 Conclusie

In de vorige sectie is Desai's algoritme vergeleken met de power methode en is ook onderzocht welke factoren van invloed zijn op de convergentiesnelheid van Desai's algoritme. We zullen nu kort alle conclusies van de simulaties op een rij zetten.
Als eerste werd gekeken naar de invloed van de steekproefgrootte op de convergentiesnelheid op basis van 50 matrices. Hieruit bleek dat een grotere steekproefgrootte niet voor een snellere convergentie zorgt. Het aantal iteraties dat nodig was om de fout met een factor 10 te verkleinen nam af, maar de hoeveelheid rekentijd die daarbij nodig was, nam in verhouding veel meer toe. Ook werd de power methode toegepast op deze matrices. Hierbij werd duidelijk dat de rekentijd bij de power methode significant kleiner was dan die bij Desai's algoritme en dat de power methode dan ook de voorkeur krijgt.
Vervolgens is de invloed van de terugkeertoestand k onderzocht voor een aantal specifiek geconstrueerde testmatrices. De terugkeertoestand k in Desai's algoritme heeft op basis van deze matrices een zeer grote invloed op de convergentiefactor ρ. Hierbij is de convergentiesnelheid oplopend van de toestand behorende bij de kleinste kolomsom naar de toestand behorende bij de grootste kolomsom van de matrix.
Na de terugkeertoestand is de invloed van de een-na-grootste eigenwaarde onderzocht. De conclusie is dat voor Desai's algoritme de een-na-grootste eigenwaarde geen invloed heeft op de convergentiesnelheid.
Ook is de beginschatting u_0 onderzocht voor invloed op de convergentiesnelheid. Ook hier bleek dat de convergentiesnelheid in Desai's algoritme niet significant werd beïnvloed voor verschillende beginschattingen.
De belangrijkste conclusie op basis van deze simulaties is als volgt: de terugkeertoestand k heeft de grootste invloed op de convergentiesnelheid van Desai's algoritme en de power methode krijgt op basis van deze resultaten de voorkeur boven Desai's algoritme. Dit laatste geeft dan ook antwoord op de hoofdvraag die gesteld is in sectie 1.

11 Discussie

• De gebruikte testmatrices
Om de convergentiesnelheid van Desai's algoritme te onderzoeken, zijn een aantal testmatrices geconstrueerd met behulp van het algoritme uit het artikel van Lin (2015). Dit resulteerde in symmetrische matrices met (reële) eigenwaarden die vooraf gekozen konden worden. Voor het simuleren zijn natuurlijk ook nog vele andere type matrices geschikt. Ook in het type matrix kan eventueel nog een groot verschil zitten in convergentiesnelheid. Een goed vervolgonderzoek zou dan ook zijn om Desai's algoritme toe te passen op meerdere type matrices.

• Implementatie Desai's algoritme in MATLAB
De code van Desai's algoritme is zelf ontworpen en vervolgens geïmplementeerd in de programmeertaal MATLAB. Door het eigen ontwerp is het mogelijk dat het algoritme niet optimaal geïmplementeerd is. Dit kan leiden tot meer rekentijd dan nodig is om tot de schattingen te komen. Ook is het mogelijk dat in een andere programmeertaal (bijvoorbeeld Python of C++) het algoritme een snellere rekentijd kan behalen.

12 Referenties

Desai, P.Y. (2001). *Adaptive Monte Carlo methods for solving eigenvalue problems*. Ann Arbor, Michigan, Verenigde Staten: Bell Howell information and Learning

Glynn, P.W. (1996). Numerical computation of large deviations exponents via simulation.*Allerton Conference on Communication, Control, and Computing*, 790-797.

Grimmett, G., Welsh, D. (2014) *Probabilty, an introduction (Second Edition)* Oxford. Oxford University Press

Desai P.Y., Glynn, P.(2001). A Markov chain perspective on adaptive Monte Carlo algorithms. Proceedings of the 2001 Winter Simulation Conference. 379-384.

Madras, N. (2002). *Lectures on Monte Carlo Methods*. Providence. American Mathematical Society

Minc, H. (1988). *Nonnegative Matrices*, Verenigde Staten, New York: a Wiley Interscience Publication.

Lin, M. M. (2015). An algorithm for constructing nonnegative matrices with prescribed real eigenvalues. Elsevier, Applied Mathematics and Computation, 256, 582–590.

Edixhoven, B. (2013, 4 november). Galois theory and the Abel-Ruffini theorem. Geraadpleegd op 17 juli 2019, van http://pub.math.leidenuniv.nl/ edixhovensj/talks/2013/indonesia/yogya/workshop.pdf

A Appendix

A.1 Definitie irreducibele matrix

Laat A een vierkante, niet-negatieve matrix zijn van dimensie d. Dan is A irreducibel als er voor alle $i, j \in \{1, 2, .., d\}$ een $n \in \mathbb{N}$ bestaat zodanig dat $A^n(i, j) > 0$.

Indien A een stochastische matrix is, kan deze matrix gezien worden als een overgangsmatrix behorende bij een Markovketen. Uit de definitie hierboven volgt dan dat als A ook irreducibel is, dat dit voor de Markovketen betekent dat vanuit elke toestand ieder andere toestand bereikt kan worden (al dan niet in meerdere stappen).

A.2 Definitie (sub)stochastische matrix

Laat P een $d \times d$ matrix zijn met niet-negatieve elementen $P(i, j)$ voor $i, j = 1, ..., d$.
Dan wordt P een **substochastische** matrix genoemd als geldt dat

$$\sum_{j=1}^{d} P(i, j) \leq 1 \qquad \forall i = 1, ..., d$$

en een **stochastische** matrix als

$$\sum_{j=1}^{d} P(i, j) = 1 \qquad \forall i = 1, ..., d$$

A.3 Achtergrond van de aanname uit sectie 4

In sectie 4 wordt een belangrijke aanname gebruikt om uiteindelijk tot de vergelijkingen (6) en (7) te komen. Het ontstaan van deze aanname zal in deze sectie worden beschreven.
Laat $Y = (Y_n, n \geq 0)$ de Markovketen zijn zoals die gedefinieerd is in sectie 4. Verder is

$$\tau = \inf\{n \geq 1 : Y_n = k\}$$

de eerste keer dat de Markovketen Y in de terugkeertoestand k terecht komt en

$$T = \inf\{n \geq 0 : Y_n = \Delta\}$$

het eerste tijdstip dat Y in het kerkhof belandt.
Definieer $P_i(\cdot) := P(\cdot | Y_0 = i)$. De theorie voor grote afwijkingen (in het Engels ook wel bekend als large deviatons theory) geeft ons dat er een $\lambda \geq 0$ bestaat zodat voor $i, j \in S$

$$\frac{1}{n} \log \Big(\mathbb{P}_k(Y_n = j, T > n) \Big) \to -\lambda$$

voor $n \to \infty$.
Dus voor grote n krijgen we de volgende benadering

$$\mathbb{P}_k(Y_n = j, T > n) \approx e^{-\lambda n} \tag{20}$$

De simultane verdelingsfunctie uit vergelijking (20) kan nu als volgt worden genoteerd:

$$\begin{aligned}\mathbb{P}_k(Y_n = j, T > n) &= \sum_{r=1}^{\infty} \mathbb{P}_k(Y_n = j, T > n, \tau = r) \\ &= \sum_{r=1}^{n} \mathbb{P}_k(Y_n = j, T > n, \tau = r) + \sum_{r=n+1}^{\infty} \mathbb{P}_k(Y_n = j, T > n, \tau = r)\end{aligned}$$

In de tweede term aan de rechterkant merken we op dat de sommatie over $T > n, \tau = r$ kan worden vervangen door de uitdrukking $T \wedge \tau > n$ (waarbij $T \wedge \tau := min(T, \tau)$). Dus ontstaat het volgende:

$$\mathbb{P}_k(Y_n = j, T > n) = \sum_{r=1}^{n} \mathbb{P}_k(Y_n = j, T > n, \tau = r) + \mathbb{P}_k(Y_n = j, T \wedge \tau > n) \tag{21}$$

Nu bekijken we de eerste term aan de rechterzijde van het gelijkteken:

$$\sum_{r=1}^{n} \mathbb{P}_k(Y_n = j, T > n, \tau = r). \tag{22}$$

Voor deze term geldt dat τ waarden aanneemt in $\{1, ..., n\}$. De doorsnede van de gebeurtenissen $\{T > n\}$ en $\{T > \tau\}$ geeft dan $\{T > n\} \cap \{T > \tau\} = \{T > n\}$. We kunnen dus de gebeurtenis $\{T > \tau\}$ bij de simultane verdelingsfunctie plaatsen zonder extra informatie toe te voegen. Dit geeft het volgende resultaat:

$$\begin{aligned}\sum_{r=1}^{n} \mathbb{P}_k(Y_n = j, T > n, \tau = r) &= \sum_{r=1}^{n} \mathbb{P}_k(Y_n = j, T > n, \tau = r, T > \tau) \\ &= \sum_{r=1}^{n} \mathbb{P}_k(Y_n = j, T > n | \tau = r, T > \tau) \mathbb{P}_k(\tau = r, T > \tau)\end{aligned}$$

Om de Markov-eigenschap op een goede manier toe te passen op de conditionele kans hierboven zullen de gebeurtenissen $\{\tau = r\}$ en $\{T > n\}$ worden uitgeschreven in termen van de Markovketen Y:

$$\begin{aligned}\{\tau = r\} &= \{Y_r = k\} \cap \left(\bigcap_{0<m<r} \{Y_m \neq k\} \right) \\ \{T > n\} &= \bigcap_{0 \leq m \leq n} \{Y_m \neq \Delta\}\end{aligned}$$

Dan kunnen we de conditionele kans als volgt herschrijven:

$$\mathbb{P}_k(Y_n = j, T > n | \tau = r, T > \tau) = \mathbb{P}_k\left((Y_n = j, \bigcap_{0\le m\le n} Y_m \neq \Delta \middle| Y_r = k, \bigcap_{0\le m\le n} Y_m \neq \Delta\right)$$

Nu geldt dat $Y_0 = k$ en $Y_r = k$. Aangezien de geschiedenis va de Markovketen er niet meer toe doet, kan de gebeurtenis $Y_r = k$ worden gezien als het moment dat het proces weer opnieuw begint in k. Merk wel op dat het bereiken van tijdstip n van de Markovketen dan nog maar $n - r$ stappen bedraagt. Deze translatie van n naar $n - r$ toepassen geeft dan het volgende:

$$\begin{aligned}\mathbb{P}_k\left((Y_n = j, \bigcap_{0\le m\le n} Y_m \neq \Delta \middle| Y_r = k, \bigcap_{0\le m\le n} Y_m \neq \Delta\right) &= \mathbb{P}_k\left(Y_{n-r} = j, \bigcap_{0\le m\le n-r} Y_m \neq \Delta\right) \\ &= \mathbb{P}_k\Big(Y_{n-r} = j, T > n - r\Big)\end{aligned}$$

Alles bij elkaar kan (22) geschreven worden als

$$\sum_{r=1}^{n} \mathbb{P}_k(Y_n = j, T > n, \tau = r) = \sum_{r=1}^{n} \mathbb{P}_k(Y_{n-r} = j, T > n - r)\mathbb{P}_k(\tau = r, T > \tau)$$

en ziet vergelijking (21) er als volgt uit:

$$\mathbb{P}_k(Y_n = j, T > n) = \sum_{r=1}^{n} \mathbb{P}_k(Y_{n-r} = j, T > n-r)\mathbb{P}_k(\tau = r, T > \tau) + \mathbb{P}_k(Y_n = j, T \wedge \tau > n).$$

Bovenstaande vergelijking kan nu worden genoteerd als

$$a(n) = b(n) + \sum_{r=1}^{n} a(n - r)g(n),$$

waarbij

$$\begin{aligned} a(n) &= \mathbb{P}_k(Y_n = j, T > n), \\ b(n) &= \mathbb{P}_k(Y_n = j, T \wedge \tau > n), \\ g(n) &= \mathbb{P}_k(\tau = n, T > \tau). \end{aligned}$$

Een vergelijking voor $a(n)$ van deze vorm wordt een zogeheten vernieuwingsvergelijking genoemd. Voor het oplossen van dit soort vergelijkingen wordt vaak beroep gedaan op de volgende aanname:

Er bestaat $\theta \geq 0$ zodanig dat

$$\sum_{n=1}^{\infty} e^{\theta n} g(n) = 1$$

Uit de definitie van $g(n)$ volgt dat dit ook schrijven is als

$$\mathbb{E}_k(e^{\theta\tau}; T > \tau) := \mathbb{E}_k(e^{\theta\tau} \cdot \mathbf{1}_{\{\tau<T\}}) = 1$$

en dit is de aanname die in sectie 4 gebruikt wordt om uiteindelijk tot Desai's algoritme te komen (waarbij $\beta_* = e^{-\theta}$).

A.4 u^* is een vast punt van Desai's algoritme

Laat u^* de unieke eigenvector zijn van de matrix P waarvoor geldt dat $u^*(k) = 1$. Dan geldt, als $X_0 = u^*$, dat $X_1 = u^*$, $X_2 = u^*$ enzovoort.

Bewijs
Er geldt $X_0 = u^*$ en $u^*(k) = 1$. Voor $i, j \in S$ geldt dan het volgende:

$$Q_{u^*}(i,j) = \frac{P(i,j)u^*(j)}{\sum_{k\in S} P(i,k)u^*(k)} = \frac{P(i,j)u^*(j)}{\lambda u^*(i)}$$

Dit geeft de volgende uitdrukking voor $W(i,\beta)$:

$$\begin{aligned}
W(i,\beta) &= \beta^{-\tau} \prod_{j=0}^{\tau-1} \frac{P(Y_j, Y_{j+1})}{Q_{u^*}(Y_j, Y_{j+1})} \\
&= \beta^{-\tau} \prod_{j=0}^{\tau-1} \frac{\lambda u^*(Y_j)}{u^*(Y_{j+1})} \\
&= \beta^{-\tau} \cdot \lambda^{\tau} \cdot \frac{u^*(Y_0)}{u^*(Y_\tau)} \\
&= \Big(\frac{\beta}{\lambda}\Big)^{-\tau} \cdot \frac{u^*(Y_0)}{u^*(k)} \\
&= \Big(\frac{\beta}{\lambda}\Big)^{-\tau} u^*(Y_0) \qquad (u^*(k) = 1)
\end{aligned}$$

Aangezien $u^*(k) = 1$, volgt dat de empirische oplossing λ_n wordt bepaald door het oplossen van

$$\tilde{W}(k,\lambda_n) = \frac{1}{n}\sum_{l=1}^{n} W(k,\lambda_n) = \frac{1}{n}\sum_{l=1}^{n} \Big(\frac{\lambda_n}{\lambda}\Big)^{-\tau_l} = 1.$$

Deze vergelijking oplossen geeft dan dat λ_n gelijk is aan λ. Dus voor de stochasten $W(i,\beta)$ geldt het volgende:

$$\begin{aligned}
W(i,\lambda_n) &= \Big(\frac{\lambda_n}{\lambda}\Big)^{-\tau} u^*(Y_0) \\
&= u^*(Y_0)
\end{aligned}$$

Dus voor iedere begintoestand $i \in S$ zal $W(i,\lambda_n)$ bijna zeker gelijk zijn aan $u^*(i)$, waaruit volgt dat de volgende beste schatting X_1 weer gelijk is aan u^*. $\square$

A.5 De wet van de totale variantie

Laat X en Y twee stochastische variabelen zijn gedefinieerd op dezelfde kansruimte en waarbij de variantie van Y eindig is. Dan geldt het volgende:

$$Var(Y) = \mathbb{E}(Var(Y|X)) + Var(\mathbb{E}(Y|X))$$

Bewijs
Voor de variantie van Y geldt het volgende:

$$\begin{aligned} Var(Y) &= \mathbb{E}(Y^2) - \mathbb{E}(Y)^2 \\ &= \mathbb{E}(\mathbb{E}(Y^2|X)) - (\mathbb{E}(\mathbb{E}(Y|X)))^2 \\ &= \mathbb{E}(Var(Y|X) + \mathbb{E}(Y|X)^2) - (\mathbb{E}(\mathbb{E}(Y|X)))^2 \\ &= \mathbb{E}(Var(Y|X)) + \Big(\mathbb{E}(\mathbb{E}(Y|X))^2) - (\mathbb{E}(\mathbb{E}(Y|X)))^2\Big) \\ &= \mathbb{E}(Var(Y|X)) + Var(\mathbb{E}(Y|X)) \end{aligned}$$

□

A.6 Het interval van de Perron-Frobenius eigenwaarde λ

Voor de gevraagde Perron-Frobenius eigenwaarde λ kan aan de hand van de bijbehorende matrix een interval worden bepaald waar deze zich in ieder geval moet bevinden. De stelling en het bewijs (gehaald uit het proefschrift van Desai (2001)) zullen nu geformuleerd worden.

Laat P een (sub)stochastische matrix zijn. Het interval waar de Perron-Frobenius eigenwaarde λ zich bevindt wordt gegeven door:

$$\min_i \sum_j P(i,j) \leq \lambda \leq \max_i \sum_j P(i,j)$$

Met andere woorden, de minimale rijsom van de matrix P vormt een ondergrens en de maximale rijsom vormt een bovengrens voor de Perron-Frobenius eigenwaarde λ.

Bewijs
Definieer u^* als de eigenvector behorende bij de eigenwaarde λ. Laat $u_{min} := min_j u^*(j)$, de kleinste component van de eigenvector u^* en laat i_{min} de bijbehorende subscript zijn. Dat geeft dan $u_{min} = u^*(i_{min})$. Dit geeft het volgende resultaat:

$$\begin{aligned} \sum_j P(i_{min}, j)u^*(j) &= \lambda u^*(i_{min}) \\ &\geq \sum_j P(i_{min}, j)u^*(i_{min}) \end{aligned}$$

Dus $\lambda \geq \sum_j P(i_{min}, j) \geq min_i \sum_j P(i, j)$. De minimale rijsom van de matrix P vormt dus een ondergrens voor de eigenwaarde λ. Voor de bovengrens geldt een soortgelijke berekening. Definieer $u_{max} := max_j u^*(j)$, de grootste component van de eigenvector u^* en laat i_{max} de bijbehorende subscript zijn. Dat geeft dan $u_{max} = u^*(i_{max})$. Dan geldt:

$$\begin{aligned}\sum_j P(i_{max}, j)u^*(j) &= \lambda u^*(i_{max}) \\ &\leq \sum_j P(i_{max}, j)u^*(i_{max})\end{aligned}$$

Dus $\lambda \leq \sum_j P(i_{max}, j) \leq max_i \sum_j P(i, j)$. De maximale rijsom van de matrix P vormt dus een bovengrens voor de eigenwaarde λ.

□

A.7 Matrix Q_u is stochastisch

Laat $P = (P(i, j) : i, j \in S)$ een (sub)stochastische matrix zijn en definieer de matrix Q_u als in vergelijking (11). Dan is de matrix Q_u een stochastische matrix.

Bewijs
Voor $i \in S$ geldt dat een willekeurige rijsom van de matrix Q_u wordt gegeven door:

$$\begin{aligned}\sum_j Q_u(i, j) &= \sum_{j \in S} Q_u(i, j) + \sum_{j \in \Delta} Q_u(i, j) \\ &= \sum_{j \in S} \left(\frac{P(i, j)u(j)}{\sum_{k \in S} P(i, k)u(k)} \right) + 0 \\ &= \frac{\sum_{j \in S} P(i, j)u(j)}{\sum_{k \in S} P(i, k)u(k)} \\ &= 1\end{aligned}$$

Voor $i \in \Delta$ geldt voor de rijsom het volgende:

$$\begin{aligned}\sum_j Q_u(i, j) &= \sum_{j \in S} Q_u(i, j) + \sum_{j \in \Delta} Q_u(i, j) \\ &= 0 + 1 \\ &= 1\end{aligned}$$

Dus de matrix $Q_u(i, j)$ is een stochastische matrix. □

B MATLAB code Desai's algoritme

In deze sectie zal de (zelf ontworpen) MATLAB code van Desai's algoritme worden getoond waarmee alle simulaties zijn uitgevoerd. Hierbij zal voor iedere stuk code een kleine toelichting worden gegeven wat zich erin bevindt.

Voor het bepalen van de overgangsmatrix Q_u is de volgende code gebruikt:

```
%Berekenen matrix Q

function Q_u = compute_Q(P,u)

[rows,columns]=size(P);
for i=1:rows
    for j=1:columns
Q(i,j) = (P(i,j)*u(j,end))/(P(i,:)*u(:,end)) ;
    end
end
Q(rows+1,:) = zeros(1,columns);
Q(:,columns+1)=[zeros((rows),1); 1];
end
```

Voor het simuleren van de Markovketen Y onder de overgangsmatrix Q_u en het bepalen van de waarde van τ is de volgende code gebruikt

```
function [tau,States] = compute_tau_2(Q,return_state,
   initial_state)
   [rows, columns]=size(Q);
   C0 = cumsum(Q,2);
   States(1) = initial_state;
   i = 1;
   if return_state == initial_state %Als de
      begintoestand gelijk is aan de 'return state'
       States(i+1)= (rows + 1) - sum(rand <= C0(States
          (i),:));
       i = i+1;
   end

while States(i)~= rows & States(i)~= return_state  %
   Zolang de 'return state' of het 'kerkhof' niet
   bereikt is
      States(i+1)= (rows + 1) - sum(rand <= C0(States(
         i),:));
      i = i+1;
end
```

```
if States(i) == rows %Als je in de 'cemetry state'
    komt
     tau = 0;

else %Als je in de 'return state' komt
     tau = length(States)-1;
end
end
```

Vervolgens worden de gesimuleerde toestanden (dus de vector 'States' uit de vorige code) gebruikt om de likelihood ratio L_τ te berekenen met deze code:

```
% Berekenen product ratio P/Q

function Is = importance_sampling(P,Q,tau,states)

if tau ==0 %Als de 'cemetery state' eerder wordt
    bereikt
     gewichten = 0;

else % Als de 'return state' eerder wordt bereikt

     for i=1:tau %Bereken voor iedere overgang de ratio
         P/Q
         gewichten(i)=P(states(i),states(i+1))/Q(states
            (i),states(i+1));
     end
end

% Neem het product van alle ratios P/Q
Is = prod(gewichten);
end
```

Voor het benaderen van de eigenwaarde (aan de hand van de bisectie methode) is vervolgens de volgende code gebruikt:

```
function p =compute_lambda(P,Q,return_state,
    interval_eigenvalue,n)
f =@(x)0;
for i=1:n

[tau,states] = compute_tau_2(Q,return_state,
    return_state);
Is = importance_sampling(P,Q,tau,states);

g =@(x) x.^(-tau)*Is;
f =@(x) g(x) + f(x);
end
f =@(x)(1/n)*f(x)-1;
a= interval_eigenvalue(1);
b =interval_eigenvalue(2);
if f(a)*f(b)>0
    p = b ;
else
    p = (a + b)/2;
    err = abs(f(p));
    while err > 1e-14
   if f(a)*f(p)<0
       b = p;
   else
       a = p;
   end
    p = (a + b)/2;
   err = abs(f(p));
    end
end
end
```

Zodra de schatting van de eigenwaarde is berekend, worden de componenten van de eigenvector benaderd met het volgende stuk code:

```
function [u,w,VKF] = compute_u(P,Q,beta,return_state,n
    )
w = [];
% Simuleer voor gegeven begintoestand
for j=1:size(P,1)
for i=1:n
[tau,states] = compute_tau_2(Q,return_state,j);
```

```
Is = importance_sampling(P,Q,tau,states);
R(j,i) = beta^(-tau)*(tau>0)*Is;
w = [w,Is];
end
end
% Neem gemiddelde van alle rijen van R
u = mean(R,2);

% Berekenen van verwachte kwadratische fout
standard_deviation_component = std(R,0,2);
standard_error_component =
   standard_deviation_component/sqrt(k);
VKF = sum(standard_error_component.^2);
end
```